teradata.

# Master Data Management

## Reference Guide

Release 4.8.0
B035-0000-9709
December 2022

# Copyrights or Trademarks

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see Section : “Teradata Trademark and Trademark Attributions.”

## Product Safety

| Safety Type | Description |
|---|---|
| | Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury. |
| | Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury. |
| | Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury. |

## Warranty Disclaimer

**Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.**

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

## Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: docs@teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

# Teradata Trademark and Trademark Attributions

Teradata, BYNET, Claraview, Covalent, DecisionCast, IntelliBase, IntelliCloud, IntelliFlex, IntelliSphere, nPath, QueryGrid, SQL-MapReduce, Stacki, "Teradata" logo, Teradata Analytics Platform, Teradata Decision Experts, "Teradata Labs" logo, Teradata ServiceConnect, and Teradata Vantage are trademarks or registered trademarks of Teradata Corporation or its affiliates in the United States and other countries.

Adaptec and SCSISelect are trademarks or registered trademarks of Adaptec, Inc.

Amazon Web Services, AWS, Amazon Elastic Compute Cloud, Amazon EC2, Amazon Simple Storage Service, Amazon S3, AWS CloudFormation, and AWS Marketplace are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.

AMD Opteron and Opteron are trademarks of Advanced Micro Devices, Inc.

Apache, Apache Avro, Apache Hadoop, Apache Hive, Hadoop, and the yellow elephant logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries.

Apple, Mac, and OS X all are registered trademarks of Apple Inc.

Axeda is a registered trademark of Axeda Corporation. Axeda Agents, Axeda Applications, Axeda Policy Manager, Axeda Enterprise, Axeda Access, Axeda Software Management, Axeda Service, Axeda ServiceLink, and Firewall-Friendly are trademarks and Maximum Results and Maximum Support are servicemarks of Axeda Corporation.

CENTOS is a trademark of Red Hat, Inc., registered in the U.S. and other countries.

Cloudera and CDH are trademarks or registered trademarks of Cloudera Inc. in the United States, and in jurisdictions throughout the world.

Data Domain, EMC, PowerPath, SRDF, and Symmetrix are either registered trademarks or trademarks of EMC Corporation in the United States and/or other countries.

GoldenGate is a trademark of Oracle.

Hewlett-Packard and HP are registered trademarks of Hewlett-Packard Company.

Hortonworks, the Hortonworks logo and other Hortonworks trademarks are trademarks of Hortonworks Inc. in the United States and other countries.

Intel, Pentium, and XEON are registered trademarks of Intel Corporation.

IBM, CICS, RACF, Tivoli, IBM Spectrum Protect, and z/OS are trademarks or registered trademarks of International Business Machines Corporation.

Linux is a registered trademark of Linus Torvalds.

LSI is a registered trademark of LSI Corporation.

Microsoft, Azure, Active Directory, Windows, Windows NT, and Windows Server are registered trademarks of Microsoft Corporation in the United States and other countries.

NetVault is a trademark of Quest Software, Inc.

Novell and SUSE are registered trademarks of Novell, Inc., in the United States and other countries.

Oracle, OpenJDK, Java, and Solaris are trademarks or registered trademarks of Oracle and/or its affiliates.

QLogic and SANbox are trademarks or registered trademarks of QLogic Corporation.

Quantum and the Quantum logo are trademarks of Quantum Corporation, registered in the U.S.A. and other countries.

Red Hat is a trademark of Red Hat, Inc., registered in the U.S. and other countries. Used under license.

SAP is the trademark or registered trademark of SAP AG in Germany and in several other countries.

SAS and SAS/C are trademarks or registered trademarks of SAS Institute Inc.

Sentinel® is a registered trademark of SafeNet, Inc.

Simba, the Simba logo, SimbaEngine, SimbaEngine C/S, SimbaExpress and SimbaLib are registered trademarks of Simba Technologies Inc.

SPARC is a registered trademark of SPARC International, Inc.

Unicode and the Unicode logo are registered trademarkes of Unicode, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Veritas, the Veritas Logo and NetBackup are trademarks or registered trademarks of Veritas Technologies LLC or its affiliates in the U.S. and other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

# Preface

## Purpose

Welcome to Teradata's Master Data Management (MDM) Product. Teradata's Master Data Management product can be thought as the set of methods or processes and procedures used to manage, reference, and synchronize correct and consistent Master Data across an Enterprise. The Teradata MDM product can provide the user with the capability to manage, integrate and consolidate Master Data with or without having to replace existing systems. Master Data can be defined as Data that is important to the company, may be referenced in transactional data, and changes over time (hence must be managed), and is needed in multiple enterprise systems by multiple users.

Teradata's MDM product provides the business users with the capability to create and manage the data. Teradata's MDM provides the ability to create flexible business workflows that accurately reflect the specific business needs of the customer and to provide this accurate and consistent information to anyone in the enterprise. Teradata's MDM is built upon an open architecture known as Service Oriented Architecture (SOA) and the Teradata platform itself, which provides the necessary performance, scalability, and reliability attributes of a World Class Master Data Management product.

With the Teradata's MDM solution you can stage, consolidate, validate, cleanse, store, augment, cross-reference, and publish data to systems in and across your enterprise. By ensuring cross-system data consistency, Teradata's Master Data Management can enable flawless and timely execution of business processes – while leveraging existing investments and reducing the total cost of ownership to manage business critical data.

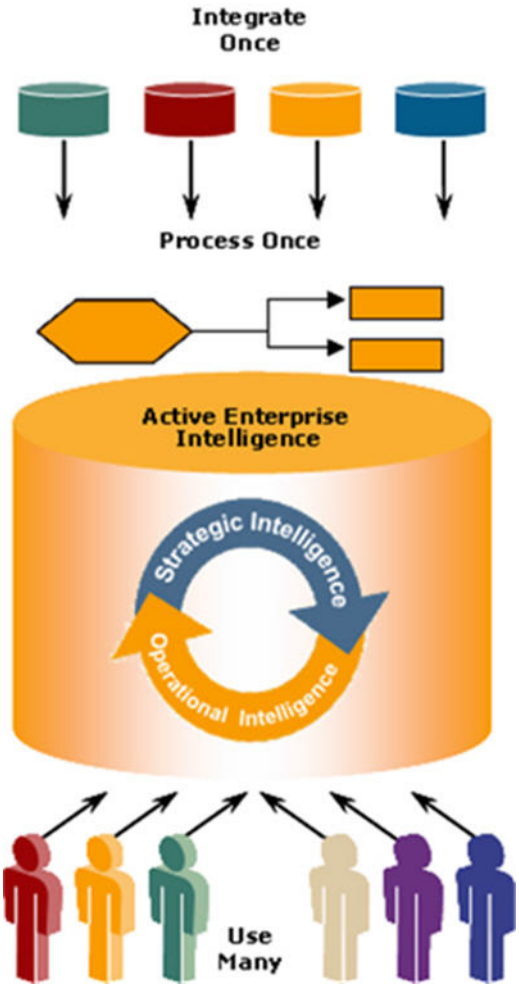
Teradata's MDM provides a framework for building business workflows by defining the necessary data in workflows via X-Docs and expressing the business logic that operates on the data via X-Rules. This framework enables the definition of business process workflows in a configurable manner with little programming effort.

Topics:


- About Teradata's Master Data Management
- About This Book
- Related Documentation
- Customer Support
- Documentation Feedback

# About Teradata's Master Data Management

Teradata's Master Data Management (MDM) helps you manage key data elements within the Enterprise Data Warehouse (EDW), within an Active Data Warehouse (ADW), as well as, across various enterprise systems and geographies.




Teradata's MDM enables this with the following high-level key features:

- Data Staging for Loading, Cleansing, and Validation.
- Master Data Life-Cycle Management for Data Maintenance and Business Specific Workflows and Processes.
- Publishing, Versioning, and Cross Referencing of Master Data.
- Data Model Extensibility and Integration with the Enterprise Data Model.
- Hierarchy Management.

# About This Book

The intention of this guide is to provide the user with a ready reference for understanding:

- Syntax and usage of built-in operations and functions in MDM
- Grammar used in defining X-Rules.
- Core framework services like Timer, Upload, and Validation etc.

The chapter on X-Documents servers as a reference and helps user to understand the data models created before the visual Data Model was introduced in MDM Studio.

This document is intended to provide the overview and recommended approach and uses of Teradata's MDM capabilities when Designing, Building, and Deploying an MDM Application. The reference guide is not intended to replace or repeat the detail information located in the various user guides, but rather reference them, as appropriate, from the overall life-cycle and development of a MDM application and focuses on the part dealing with the syntax and grammar for business rules.

## Target Audience

This guide is intended primarily for those individuals who are planning, participating, and responsible for Designing, or Building, or Deploying a MDM application. The Teradata's MDM product is a Teradata optimized product and knowledge of Teradata capabilities and architecture are embraced within this developers guide.

## What You Should Know

This guide assumes that the audience is familiar with basic development practices of a software application. In addition, the audience should have some familiarity with Services oriented architectures and Web Services. The Teradata's MDM product provides the capability to develop and deploy a business process as a workflow, therefore, in the design of the MDM application any experience in business process analysis and workflow analysis would be beneficial. The data model used by the Teradata's MDM product is aligned to the utilization of Data Models in the Teradata EDW/ADW context, therefore, any experience with Teradata data model implementations would be beneficial.

## Document Structure

This book contains the following chapters:

- "Section AUI Reference" lists various chapters on UI reference.
- "Section BService Reference" lists various chpaters on service reference.
- "Appendix AA  Frequently Asked Questions" lists the FAQs.

## Changes to This Book

The following changes were made to this book in support of the current release. For a complete list of changes to the product, refer *MDM Platform Release Definition* associated with this release.

| Date and Release | Description |
|---|---|
| August 2013 3.3 | Included Section : “REMOVE_NODES” |
| April 2014 3.3.1 | Included Section : “Data List” and Section : “Placeholder Attribute” |
| November 2014 3.4 | Included Appendix W: “Parent-Child Validation.” |
| January 2015 3.4.0.1 | Included Section : “MDM Pop-ups” in Appendix A: “Using the PGL Example Project.” |
| February 2015 3.4.1 | Included Chapter 3: “New HTML5 Compliant UI.” |
| November 2015 3.5 | Included PGL framework architecture in Chapter 1: “Page Layout.” Replaced Appendix Hierarchy viewer entry and exit with Hierarchy Viewer - Custom Workflow. Included the <iframe> control. |
| March 2015 3.5.1 | Removed the Refresh attribute from optional attributes description under Button section. Included EXEC_PARALLEL Command Details. |
| June 2016 3.5.2 | Included Appendix X: “Dashboard Technology Implementation.” |
| November 2016 3.5.3 | Version details updated. |
| August 2017 4.0.0 | DO_DB_Persist Framework updated in Chapter 9: “X-Operations” |
| September 2017 4.0.1 | Version details updated. |
| December 2017 4.1.0 | Version details updated. |
| June 2018 4.2.0 | Version details updated. |
| April 2019 4.3.0 | DO_DB_Persist command updated. |
| November 2019 4.4.0 | Removed CRDM APIs Appendix and added Chapter 11: “MDM JSON Framework” |
| June 2020 4.5.0 | Version details updated. |

| Date and Release | Description |
|---|---|
| March 2021<br>4.6.0 | Added Chapter 12: “PGL UI Material Design Style.”Appendix Y: “Angular UI for Custom Workflows.” |
| March 2022<br>4.7.0 | Added new Appendix for Error handling for Left navigation when shared services are unavailable. |
| November 2022, 4.8.0 | Release Version Updated<br>Updated “Appendix Y: Angular UI for Custom Workflows” |

# Related Documentation

For more information on MDM, refer the following documents:

- *Master Data Management Studio User Guide*
  (Master Data Management 4.8.0 Studio User Guide.pdf)
- *Master Data Management Release Definition*
  (Master Data Management 4.8.0 Release Definition.pdf)
- *Master Data Management Server Guide*
  (Master Data Management 4.8.0 Server Guide.pdf)
- *Master Data Management Installation Guide*
  (Master Data Management 4.8.0 Installation Guide.pdf)
- *Master Data Management Developer Guide*
  (Master Data Management 4.8.0 Developer Guide.pdf)

The above Teradata documents are available at: https://docs.teradata.com

## To Read The Documentation

To read the .pdf files, you must have Adobe Acrobat Reader, version 4.0 or higher. If you do not have Acrobat Reader on your machine, you can download it from Adobe’s Web site at http://www.adobe.com.

# Customer Support

Customer support is available at the Teradata customer support Web site (https://access.teradata.com), where you can:

- Request shipment of software.
- Download software documentation.
- Submit new issues or cases.
- Track the status of current issues or cases.

# Documentation Feedback

Please share your thoughts and ideas:

- Send feedback to docs@teradata.com.
- Navigate to https://teradata-documentation.ideas.aha.io/ideas/new and provide your ideas.

# Table of Contents


Purpose . . . . . iii
About Teradata's Master Data Management . . . . . iv
About This Book . . . . . v
Target Audience . . . . . v
What You Should Know . . . . . v
Document Structure . . . . . v
Changes to This Book . . . . . vi
Related Documentation . . . . . vii
To Read The Documentation . . . . . vii
Customer Support . . . . . vii
Documentation Feedback . . . . . viii

**Section A: UI Reference** . . . . . 1

**Chapter 1: Page Layout** . . . . . 2

Introduction . . . . . 2
Page Layout . . . . . 2
Getting to Know PGL . . . . . 3
Object Association . . . . . 3
View Hierarchies . . . . . 4
Document Structure . . . . . 4
Element Realization . . . . . 4
Element Attributes . . . . . 4
PGL Framework Architecture . . . . . 4
PGL Generic Notes . . . . . 8

**Chapter 2: Getting Started** . . . . . 9

Introduction . . . . . 9
The Hello World Example . . . . . 9

## Chapter 3: New HTML5 Compliant UI . . . . . 11

Introduction . . . . . 11
Major Framework Changes . . . . . 11
HTML/CSS Changes . . . . . 11
Removal of Frames . . . . . 12
Load Content Click of Links . . . . . 14
Form Submission (Load Content on Click of Buttons) . . . . . 15
File Downloads . . . . . 16
File Uploads . . . . . 17
Hide Left Navigator . . . . . 18
History Maintenance . . . . . 19
Minify JavaScript . . . . . 20
Scroll Bar Maintenance . . . . . 20
PGL Changes . . . . . 24

## Chapter 4: Basic Controls . . . . . 28

Introduction . . . . . 29
Container . . . . . 30
Syntax . . . . . 30
Examples . . . . . 30
Example Data . . . . . 31
Required Attributes . . . . . 32
Optional Attributes . . . . . 32
Content . . . . . 32
Used By . . . . . 33
Step . . . . . 33
Examples . . . . . 33
Example Data . . . . . 34
Required Attributes . . . . . 35
Optional Attributes . . . . . 35
Used By . . . . . 36
Content . . . . . 36
Search and Results . . . . . 36
Syntax . . . . . 36
Examples . . . . . 36
Example Data . . . . . 40
Required Attributes . . . . . 40
Optional Attributes . . . . . 40

Used By . . . . . 41
Content . . . . . 41
Text Area . . . . . 41
Syntax . . . . . 41
Example . . . . . 42
Example Data . . . . . 42
Required Attributes . . . . . 43
Optional Attributes . . . . . 43
Content . . . . . 46
Used By . . . . . 46
Button . . . . . 46
Syntax . . . . . 46
Example . . . . . 46
Required Attributes . . . . . 48
Optional Attributes . . . . . 48
Content . . . . . 49
Used By . . . . . 49
Button Group . . . . . 49
Syntax . . . . . 49
Example . . . . . 50
Example Data . . . . . 50
Required Attributes . . . . . 50
Optional Attributes . . . . . 50
Content . . . . . 51
Used By . . . . . 51
Check Box . . . . . 51
Syntax . . . . . 51
Example . . . . . 51
Example Data . . . . . 52
Required Attributes . . . . . 53
Optional Attributes . . . . . 53
Used By . . . . . 55
Content . . . . . 55
Radio Button . . . . . 56
Syntax . . . . . 56
Example . . . . . 56
Example Data . . . . . 57
Required Attributes . . . . . 58
Optional Attributes . . . . . 58
Used By . . . . . 60
Content . . . . . 60
Display Field . . . . . 61

Syntax. . . . . 61
Example . . . . . 61
Example Data. . . . . 62
Required Attributes . . . . . 63
Optional Attributes. . . . . 63
Used By . . . . . 65
Content. . . . . 65
Chart Field . . . . . 66
Syntax. . . . . 66
Example . . . . . 66
Example Data. . . . . 66
Required Attributes . . . . . 66
Optional Attributes. . . . . 67
Used By . . . . . 67
Content. . . . . 67
Entry Field . . . . . 67
Syntax. . . . . 67
Example . . . . . 67
Example Data. . . . . 69
Required Attributes . . . . . 70
Optional Attributes. . . . . 70
Used By . . . . . 72
Content. . . . . 72
Placeholder Attribute . . . . . 72
Data List . . . . . 73
Syntax. . . . . 73
Example . . . . . 73
Example Data. . . . . 74
Required Attributes . . . . . 74
Optional Attributes. . . . . 74
Used By . . . . . 75
Content. . . . . 75
Hidden Field . . . . . 75
Syntax. . . . . 75
Example . . . . . 75
Example Data. . . . . 75
Required Attributes . . . . . 76
Optional Attributes. . . . . 76
Used By . . . . . 76
Content. . . . . 76
Field Group. . . . . 77
Syntax. . . . . 77

Examples . . . 77
Example Data . . . 80
Required Attributes . . . 80
Optional Attributes . . . 81
Used By . . . 81
Content . . . 81
Field Space . . . 81
Syntax . . . 81
Example . . . 81
Example Data . . . 82
Required Attributes . . . 82
Optional Attributes . . . 83
Used By . . . 83
Content . . . 83
Divider . . . 83
Syntax . . . 83
Example . . . 83
Example Data . . . 84
Required Attributes . . . 84
Optional Attributes . . . 84
Used By . . . 84
Content . . . 84
Drop Down . . . 84
Syntax . . . 84
Example . . . 85
Example Data . . . 85
Required Attributes . . . 88
Optional Attributes . . . 88
Used By . . . 90
Content . . . 90
Option . . . 90
Syntax . . . 90
Examples . . . 91
Example Data . . . 91
Required Attributes . . . 91
Optional Attributes . . . 92
Used By . . . 92
Content . . . 92
Horizontal Rule (Hr) . . . 92
Syntax . . . 92
Example . . . 92
Example Data . . . 93

Required Attributes . . . . . 93
Optional Attributes. . . . . 93
Used By . . . . . 93
Content . . . . . 93
Vertical Rule (Vr) . . . . . 93
Syntax. . . . . 93
Required Attributes . . . . . 93
Optional Attributes. . . . . 93
Used By . . . . . 94
Content . . . . . 94
Example Data . . . . . 94
Header. . . . . 94
Syntax. . . . . 94
Example . . . . . 94
Example Data . . . . . 94
Required Attributes . . . . . 94
Optional Attributes. . . . . 95
Used By . . . . . 95
Content . . . . . 95
Help . . . . . 95
Syntax. . . . . 95
Example . . . . . 95
Example Data . . . . . 95
Required Attributes . . . . . 96
Optional Attributes. . . . . 96
Used By . . . . . 96
Content . . . . . 96
Validation . . . . . 96
Syntax. . . . . 96
Example . . . . . 96
Example Data . . . . . 98
Required Attributes . . . . . 99
Optional Attributes. . . . . 99
Used By . . . . . 99
Content . . . . . 99
Date. . . . . 99
Syntax. . . . . 99
Example . . . . . 99
Example Data . . . . . 100
Required Attributes . . . . . 100
Optional Attributes. . . . . 100
Used By . . . . . 102

Content . . . . . 102
Compare . . . . . 102
Syntax . . . . . 102
Example Data . . . . . 102
Required Attributes . . . . . 102
Optional Attributes . . . . . 102
Used By . . . . . 103
Content . . . . . 103
Page Group . . . . . 103
Syntax . . . . . 103
Example . . . . . 103
Example Data . . . . . 104
Required Attributes . . . . . 104
Optional Attributes . . . . . 104
Used By . . . . . 105
Content . . . . . 105
Chart . . . . . 105
Syntax . . . . . 105
Example . . . . . 105
Example Data . . . . . 106
Required Attributes . . . . . 106
Optional Attributes . . . . . 107
Used By . . . . . 107
Content . . . . . 107
Root . . . . . 107
Syntax . . . . . 107
Example Data . . . . . 107
Required Attributes . . . . . 107
Optional Attributes . . . . . 107
Used By . . . . . 108
Content . . . . . 108
Custom Validator . . . . . 108
Syntax . . . . . 108
Example Data . . . . . 108
Required Attributes . . . . . 108
Optional Attributes . . . . . 108
Used By . . . . . 108
Content . . . . . 108
For Each . . . . . 109
Syntax . . . . . 109
Example Data . . . . . 109
Required Attributes . . . . . 109

Optional Attributes. . . . . 109
Used By . . . . . 109
Content . . . . . 109
Login. . . . . 109
Syntax. . . . . 109
Example . . . . . 110
Example Data. . . . . 110
Required Attributes . . . . . 110
Optional Attributes. . . . . 110
Used By . . . . . 111
Content . . . . . 111
Navigation . . . . . 111
Syntax. . . . . 111
Example . . . . . 111
Example Data. . . . . 112
Required Attributes . . . . . 113
Optional Attributes. . . . . 113
Used By . . . . . 113
Content . . . . . 113
Non-Breaking Space. . . . . 113
Syntax. . . . . 113
Example Data. . . . . 113
Required Attributes . . . . . 113
Optional Attributes. . . . . 113
Used By . . . . . 114
Content . . . . . 114
Pad . . . . . 114
Syntax. . . . . 114
Example . . . . . 114
Example Data. . . . . 115
Required Attributes . . . . . 115
Optional Attributes. . . . . 115
Used By . . . . . 116
Content . . . . . 116
Pad Item . . . . . 116
Syntax. . . . . 116
Example . . . . . 116
Example Data. . . . . 117
Required Attributes . . . . . 117
Optional Attributes. . . . . 117
Used By . . . . . 118
Content . . . . . 118

Sort . . . . . 118
Syntax. . . . . 118
Example Data. . . . . 118
Required Attributes . . . . . 118
Optional Attributes. . . . . 118
Used By . . . . . 119
Content. . . . . 119
Image . . . . . 119
Syntax. . . . . 119
Example . . . . . 119
Example Data. . . . . 120
Required Attributes . . . . . 120
Optional Attributes. . . . . 120
Used By . . . . . 120
Content. . . . . 121
Image Link Field . . . . . 121
Syntax. . . . . 121
Example . . . . . 121
Example Data. . . . . 122
Required Attributes . . . . . 122
Optional Attributes. . . . . 122
Used By . . . . . 124
Content. . . . . 124
Link. . . . . 124
Syntax. . . . . 124
Example . . . . . 125
Example Data. . . . . 125
Required Attributes . . . . . 125
Optional Attributes. . . . . 125
Used By . . . . . 126
Content. . . . . 126
Link Field . . . . . 126
Syntax. . . . . 126
Example . . . . . 126
Example Data. . . . . 127
Required Attributes . . . . . 128
Optional Attributes. . . . . 128
Used By . . . . . 130
Content. . . . . 130
Links. . . . . 130
Syntax. . . . . 130
Example . . . . . 130

Example Data . . . . . 130
Required Attributes . . . . . 130
Optional Attributes . . . . . 130
Used By . . . . . 130
Content . . . . . 130
URL . . . . . 130
Syntax . . . . . 131
Example . . . . . 131
Example Data . . . . . 131
Required Attributes . . . . . 131
Optional Attributes . . . . . 131
Used By . . . . . 131
Content . . . . . 132
UI:DIV . . . . . 132
Syntax . . . . . 132
Example . . . . . 132
Example Data . . . . . 133
Required Attributes . . . . . 133
Optional Attributes . . . . . 133
Used By . . . . . 133
Content . . . . . 133
Table . . . . . 133
Syntax . . . . . 134
Examples . . . . . 134
Example Data . . . . . 137
Required Attributes . . . . . 143
Optional Attributes . . . . . 143
Used By . . . . . 145
Content . . . . . 145
Treecell . . . . . 145
Syntax . . . . . 145
Example . . . . . 145
Example Data . . . . . 146
Required Attributes . . . . . 146
Optional Attributes . . . . . 146
Used By . . . . . 148
Content . . . . . 148
IFrame . . . . . 148
Syntax . . . . . 148
Example . . . . . 148
Example Data . . . . . 148
Required Attributes . . . . . 148

Optional Attributes. . . . . 149
Used By . . . . . 149
Content . . . . . 149

**Chapter 5: Advanced Controls** . . . . . 150

Introduction . . . . . 150
Grid . . . . . 151
Syntax . . . . . 151
Example . . . . . 151
Example Data . . . . . 152
Required Attributes . . . . . 152
Optional Attributes . . . . . 152
Used By . . . . . 152
Content . . . . . 152
Row . . . . . 152
Syntax . . . . . 152
Example . . . . . 152
Example Data . . . . . 153
Required Attributes . . . . . 153
Optional Attributes . . . . . 153
Used By . . . . . 153
Content . . . . . 153
Cell . . . . . 153
Syntax . . . . . 153
Example . . . . . 154
Example Data . . . . . 154
Required Attributes . . . . . 154
Optional Attributes . . . . . 154
Used By . . . . . 154
Content . . . . . 155
Script . . . . . 155
Syntax . . . . . 155
Example . . . . . 155
Example Data . . . . . 156
Required Attributes . . . . . 156
Optional Attributes . . . . . 156
Used By . . . . . 156
Content . . . . . 156
Choose Field . . . . . 156
Syntax . . . . . 156

Example . . . . . 157
Example Data . . . . . 158
Required Attributes . . . . . 159
Optional Attributes . . . . . 159
Used By . . . . . 161
Content . . . . . 161
When . . . . . 161
Syntax . . . . . 161
Example . . . . . 162
Example Data . . . . . 162
Required Attributes . . . . . 162
Optional Attributes . . . . . 162
Used By . . . . . 164
Content . . . . . 164
Otherwise . . . . . 164
Syntax . . . . . 164
Example Data . . . . . 164
Required Attributes . . . . . 164
Optional Attributes . . . . . 165
Used By . . . . . 166
Content . . . . . 166
Dynamic PGL . . . . . 167
Syntax . . . . . 167
Example . . . . . 167
Example Data . . . . . 167
Required Attributes . . . . . 169
Optional Attributes . . . . . 169
Used By . . . . . 169
Content . . . . . 169
Excel Pivot . . . . . 169
Syntax . . . . . 170
Example . . . . . 170
Example Data . . . . . 170
Required Attributes . . . . . 172
Optional Attributes . . . . . 172
Used By . . . . . 172
Content . . . . . 172
Horizontal Tree . . . . . 172
Syntax . . . . . 173
Example . . . . . 173
Example Data . . . . . 173
Required Attributes . . . . . 175

Optional Attributes. . . . . 175
Used By . . . . . 176
Content . . . . . 176
Legend Field . . . . . 176
Syntax. . . . . 176
Example Data . . . . . 176
Required Attributes . . . . . 176
Optional Attributes. . . . . 176
Used By . . . . . 177
Content . . . . . 177
Pivot . . . . . 177
Syntax. . . . . 177
Example . . . . . 177
Example Data . . . . . 177
Required Attributes . . . . . 179
Optional Attributes. . . . . 179
Used By . . . . . 180
Content . . . . . 180
Pivot Data . . . . . 180
Syntax. . . . . 180
Example Data . . . . . 180
Required Attributes . . . . . 180
Optional Attributes. . . . . 180
Used By . . . . . 181
Content . . . . . 181
Pivot Field. . . . . 181
Syntax. . . . . 181
Example Data . . . . . 181
Required Attributes . . . . . 181
Optional Attributes. . . . . 181
Used By . . . . . 182
Content . . . . . 182
Spreadsheet . . . . . 182
Syntax. . . . . 182
Example Data . . . . . 182
Required Attributes . . . . . 183
Optional Attributes. . . . . 183
Used By . . . . . 183
Content . . . . . 183
Spreadsheet Field . . . . . 183
Syntax. . . . . 183
Example Data . . . . . 183

Required Attributes . . . . . 183
Optional Attributes. . . . . 184
Used By . . . . . 186
Content . . . . . 186
Spreadsheet Row . . . . . 186
Syntax. . . . . 186
Example Data . . . . . 186
Required Attributes . . . . . 186
Optional Attributes. . . . . 186
Used By . . . . . 187
Content . . . . . 187
Tree Node . . . . . 187
Syntax. . . . . 187
Example Data . . . . . 187
Required Attributes . . . . . 187
Optional Attributes. . . . . 187
Used By . . . . . 188
Content . . . . . 188
Vertical Tree . . . . . 188
Syntax. . . . . 188
Example . . . . . 188
Example Data . . . . . 188
Required Attributes . . . . . 190
Optional Attributes. . . . . 190
Used By . . . . . 191
Content . . . . . 191
Workbook . . . . . 191
Syntax. . . . . 191
Example Data . . . . . 192
Required Attributes . . . . . 198
Optional Attributes. . . . . 198
Used By . . . . . 198
Content . . . . . 198
Shell . . . . . 199
Syntax. . . . . 199
Example Data . . . . . 199
Required Attributes . . . . . 199
Optional Attributes. . . . . 199
Used By . . . . . 199
Content . . . . . 199
DIV . . . . . 200
Syntax. . . . . 200

Example . . . . . 200
Example Data . . . . . 201
Required Attributes . . . . . 201
Optional Attributes . . . . . 201
Used By . . . . . 201
Content . . . . . 201

**Chapter 6: MDM Elements** . . . . . 202

Master Data Management . . . . . 202
MDM Studio . . . . . 202
Run-Time Server . . . . . 203
MDM Elements . . . . . 203
X-Server . . . . . 203
X-Service . . . . . 204
X-Workflow . . . . . 204
X-Rules . . . . . 205
X-Documents . . . . . 205

**Section B: Service Reference** . . . . . 206

**Chapter 7: X-Document** . . . . . 207

Introduction . . . . . 207
Document Identifiers . . . . . 208
Surrogate Key . . . . . 210
Document Index . . . . . 210
Document Properties . . . . . 211
boolean . . . . . 212
char . . . . . 212
clob . . . . . 212
date . . . . . 212
decimal . . . . . 212
double . . . . . 212
encryptedString . . . . . 212
float . . . . . 213
int . . . . . 213
String . . . . . 213

ST_Geometry . . . 213
timestamp . . . 214
xml . . . 214
Document Links . . . 214
Foreign Key Constraints . . . 216
Overlay Documents . . . 217
Audit Trail . . . 218
Document Constraints . . . 218

**Chapter 8: x-Rules** . . . 221

Introduction . . . 221
Variables and Expressions . . . 223
Special Variables . . . 224
Condition Statements . . . 225
Simple Condition . . . 225
Compound Condition . . . 226
Business Requests . . . 227
DEFINE_METHOD . . . 227
DEFINE_NOTIFICATION . . . 227
DEFINE_PRE_CREATE . . . 227
DEFINE_POST_CREATE . . . 227
DEFINE_PRE_MODIFY . . . 227
DEFINE_POST_MODIFY . . . 227
DEFINE_INIT . . . 228
DEFINE_LISTENER . . . 228
Invoking X-Rules . . . 243
REQUEST . . . 243
SEND_NOTIFICATION . . . 244

**Chapter 9: X-Operations** . . . 246

Introduction . . . 246
X-Document Operations . . . 247
ADD_DOCUMENT . . . 247
CREATE_DOCUMENT_ID . . . 249
QUERY CRITERIA . . . 250
REMOVE_DOCUMENT . . . 251
MODIFY_DOCUMENT . . . 252

SAVE . . . . . 253
GET_DOCUMENT . . . . . 254
Expression . . . . . 258
ORDER_BY . . . . . 259
GROUP_BY . . . . . 261
QUERY_DOCUMENT_LINK . . . . . 262
NOT_EXIST_QDL . . . . . 264
SELECT + QUERY_DOCUMENT_LINK . . . . . 266
FETCH_DOCUMENT_LINK . . . . . 268
GET_DOCUMENT_LINK . . . . . 270
PROCESS_GET_DOCUMENT . . . . . 272
Smart XCommands . . . . . 274
ADD_DOC_SMART . . . . . 274
GET_DOC_SMART . . . . . 274
MODIFY_DOC_SMART . . . . . 275
REMOVE_DOC_LOGICAL . . . . . 275
SQL Operations . . . . . 276
SQL_FN . . . . . 276
EXECUTE_SQL_QUERY . . . . . 281
EXECUTE_SQL_DML . . . . . 282
EXECUTE_BATCH_SQL_DML . . . . . 283
EXECUTE_SQL_DDL . . . . . 284
EXEC_PROC . . . . . 284
Child Tags/Attributes . . . . . 285
EXEC_PARALLEL . . . . . 286
XML Manipulation Operations . . . . . 287
SET . . . . . 287
SET_PROPS . . . . . 288
TO_DOCVAR . . . . . 289
GET_TEXT . . . . . 298
SET_CHILD . . . . . 298
ADD_CHILDREN . . . . . 299
APPEND_CHILDREN . . . . . 302
REMOVE_CHILDREN . . . . . 303
REMOVE_NODES . . . . . 304
DELETE_ALL_CHILDREN . . . . . 308
COLLATE_XML . . . . . 309
MAP . . . . . 312
GROUP_DOCS . . . . . 314
MERGE_DOC . . . . . 316
MERGE_ERRORS_DOC . . . . . 318
STRING_TO_XML . . . . . 318
XML_TO_STRING . . . . . 319

CLONE_XML . . . . . 319
SORT_DOCS . . . . . 320
SORT . . . . . 324
Logical Operations . . . . . 326
IF_TEST, THEN, ELSE . . . . . 326
IF, THEN, ELSE . . . . . 328
FOR_EACH, BREAK, CONTINUE . . . . . 330
REPEAT . . . . . 331
WHILE . . . . . 332
EXIT . . . . . 334
User Interface Related Operations . . . . . 335
PAD . . . . . 335
PAD_ITEM . . . . . 335
SET_SESSION . . . . . 336
GET_SESSION . . . . . 337
FORM_SEARCH_FILTER . . . . . 337
Child Tag Attributes . . . . . 338
FORM_GET . . . . . 340
FORM_REMOVE . . . . . 342
FORM_UPDATE . . . . . 343
TAG_ERRORS . . . . . 346
DROP_DOWN_OPTIONS . . . . . 349
CREATE URL . . . . . 350
Utility Operations . . . . . 351
START_WORKFLOW . . . . . 351
PRINT/PRINTLN . . . . . 352
LOG . . . . . 353
EXECUTE_SHELL_COMMAND . . . . . 353
STORE_XML_FILE . . . . . 355
PRINT_TO_FILE . . . . . 355
GET_CURRENT_TIME . . . . . 356
SET_LOGICAL_DATE . . . . . 356
HANDLE_SYSTEM_EXCEPTION . . . . . 356
EXCEPTION . . . . . 357
DO_TRANSACTION . . . . . 359
REFRESH_FROM_CACHE . . . . . 359
EXEC_INFORMATICA . . . . . 360
EXECUTE_TRILLIUM . . . . . 362
NATIVE_FUNCTION . . . . . 366
INSTANCE_FUNCTION . . . . . 367
INSTANCE_METHOD . . . . . 368
NOTIFY . . . . . 368
EVAL_EXPR . . . . . 369

Framework Service Operations . . . . . 369
VALIDATE_DOC . . . . . 369
UPDATE_VALIDATION_RESULT . . . . . 370
DO_DB_PERSIST Framework Architecture . . . . . 371

**Chapter 10: X-Path** . . . . . 378

Introduction . . . . . 378
X-Path Basics . . . . . 379
Selecting a node . . . . . 380
Selecting a list of nodes . . . . . 381
Selecting a list of nodes based on a condition . . . . . 382
Boolean X-Path Expressions . . . . . 383
Node-Set Functions . . . . . 386
count . . . . . 387
name . . . . . 387
String Functions . . . . . 388
string . . . . . 389
concat . . . . . 390
strlen . . . . . 390
starts-with . . . . . 390
subString . . . . . 391
substringBefore . . . . . 391
substringAfter . . . . . 391
normalize-space . . . . . 392
trim . . . . . 392
upperCase . . . . . 392
lowerCase . . . . . 393
Boolean Functions . . . . . 393
true() . . . . . 394
false() . . . . . 394
Number Functions . . . . . 394
int . . . . . 395
float . . . . . 396
sum . . . . . 396
Arithmetic Functions . . . . . 396
+ (plus) . . . . . 397
- (minus) . . . . . 397
mul . . . . . 397
div . . . . . 398
mod . . . . . 398

Comparison Functions . . . . . 398
< (less than) . . . . . 398
> (greater than) . . . . . 399
<= (less than or equal to) . . . . . 399
>= (greater than or equal to) . . . . . 400
= (equals) . . . . . 400
!= (not equal) . . . . . 401
and . . . . . 401
or . . . . . 402
Time Functions . . . . . 402
date . . . . . 402
getYear . . . . . 403
getMonth . . . . . 404
getDay . . . . . 405
getHour . . . . . 407
getMinute . . . . . 408
getSecond . . . . . 409
duration . . . . . 410
incrDate . . . . . 410
datediff . . . . . 411
parseDate . . . . . 411
Java's SimpleDateFormat . . . . . 411
Date and Time Patterns . . . . . 411
Miscellaneous Functions . . . . . 414
getFirstDoc . . . . . 415
getDocument . . . . . 416
isValidValue . . . . . 417
getDBInfo . . . . . 418
getWorkingDirectory . . . . . 418
Java Functions . . . . . 418
nativeFunction . . . . . 418
instanceFunction . . . . . 419
instanceMethod . . . . . 421

**Chapter 11: MDM JSON Framework** . . . . . 423

Introduction . . . . . 423
Standards for JSON Framework . . . . . 423
X-Rules . . . . . 423
CRUD APIs . . . . . 424
XML and Equivalent JSON Request and Response . . . . . 430

Guidelines . . . . . 435

## Chapter 12: PGL UI Material Design Style . . . . . 436

Introduction . . . . . 436
PGL UI Style Improvement . . . . . 436
New PGL Version Introduced for Material Design . . . . . 436
Data Grid Improvements . . . . . 437
Forms Improvements . . . . . 443
General Guidelines . . . . . 445

## Appendix A: Using the PGL Example Project . . . . . 446

Introduction . . . . . 446
Location . . . . . 446
Workflows . . . . . 447
Basic Controls . . . . . 447
Advanced Controls . . . . . 448
Samples . . . . . 448
Running Workflows . . . . . 448
Example . . . . . 451
MDM Pop-ups . . . . . 455

## Appendix B: Hierarchy Viewer-Custom Workflow . . . . . 457

Implementation Options . . . . . 457

## Appendix C: Timer Service . . . . . 460

Introduction . . . . . 460
Deployment . . . . . 460
Timer Setup and Callback . . . . . 461
Configuration of Timer Poll Interval . . . . . 461
Timer Commands . . . . . 462
START_TIMER . . . . . 462
STOP_TIMER . . . . . 463

Timer Functions . . . . . 464
Timer Examples . . . . . 464
Guaranteed Messaging . . . . . 467
Success-failure Determination . . . . . 469

**Appendix D: Validation Services** . . . . . 472

Introduction . . . . . 472
Variables . . . . . 472
Attributes . . . . . 473
Types of Validations . . . . . 474
Types of Resolutions . . . . . 477
Structure of a Validation Specification . . . . . 478
Using Validation Framework . . . . . 479
Writing the Validation Specification . . . . . 480
Specifying the Validation Specification File . . . . . 480
Including Validation Specification File Path . . . . . 480
Invoking Validation Framework . . . . . 480

**Appendix E: Upload Services** . . . . . 485

Introduction . . . . . 485
Assisted Upload . . . . . 485
XML Based Data Upload Template . . . . . 485
unassisted Upload . . . . . 486

**Appendix F: ID Generation** . . . . . 488

Introduction . . . . . 488
TimeIdGeneration . . . . . 488
TableIdGeneration . . . . . 488
BlockIdGeneration . . . . . 489
Configuration . . . . . 489
Generating Next Id . . . . . 490

## Appendix G: Events and Request Interfaces . . . . . 491

Defining an Event . . . . . 491
Subscribing Events . . . . . 493
Defining a Listener in X-Rules . . . . . 493
Defining a Listener in Workflow . . . . . 493
Defining a Listener in Through Event Spec . . . . . 494
Raising Events . . . . . 497
Defining a Request Interface . . . . . 499

## Appendix H: Using Rules Language Project . . . . . 501

Location . . . . . 501
Generating Schema . . . . . 502
Rules . . . . . 504
Running Rules . . . . . 505

## Appendix I: X-Service APIs and WSDLs . . . . . 506

Introduction . . . . . 506
API_DOC Interface Description Language . . . . . 507
Specifying the Data Type of an Attribute . . . . . 508
Specifying that an Attribute is Optional . . . . . 509
Specifying that Tag is Optional . . . . . 509
Allowing Extra Attributes on a Tag . . . . . 509
Specifying the Order of Child Tags . . . . . 509
Allowing Extra Child Tags . . . . . 510
Allowing Repetitions of a Tag . . . . . 510
Specifying a Choice of Tags . . . . . 511
API_DOC Language Extensions . . . . . 512
Referencing Declarations in an Xdocument . . . . . 512
Referencing a Search Spec . . . . . 514
Referencing a Validation Spec . . . . . 516
Advanced Usage . . . . . 519
Referencing Other API_DOCs . . . . . 520
Generating Xservice WSDL . . . . . 522
API Access Control on MDM Services . . . . . 523
Validating Xservice APIs at Runtime . . . . . 523

Adding Custom Extensions to API_DOC . . . . . 524

## Appendix J: Data Relationship Management APIs . . . . . 526

Manage Relationship Objects . . . . . 526
Create . . . . . 526
Edit . . . . . 527
Delete . . . . . 528
Input Parameters Description . . . . . 528
Manage Filter . . . . . 529
Create . . . . . 529
Update . . . . . 529
Delete . . . . . 530
Input Parameters Description . . . . . 530
Manage Relationship . . . . . 531
Create . . . . . 531
Link Data . . . . . 532
Delete . . . . . 533
Input Parameters Description . . . . . 533
Manage Cross Reference . . . . . 533
Create . . . . . 533
Delete . . . . . 534
Input Parameters Description . . . . . 534
Hierarchy Manager Rules . . . . . 535
Create Hierarchy . . . . . 535
Delete Hierarchy . . . . . 536
Create Hierarchy Version . . . . . 536

## Appendix K: MDM Generic APIs . . . . . 537

Enterprise Model (User Security) APIs . . . . . 537
Add Organization . . . . . 537
Modify Organization . . . . . 538
Get Organization . . . . . 538
Manage Role Details . . . . . 539
Add Role . . . . . 540
Get Assignable Roles . . . . . 540
Get Roles for User . . . . . 541
Validate Role for User . . . . . 542
Validate Role . . . . . 542

Get User Role . . . . . 542
Add User . . . . . 543
Modify User . . . . . 544
Get Users . . . . . 544
Get User . . . . . 545
Validate Activity for User . . . . . 546
Add User Group . . . . . 546
Create User Group . . . . . 547
Modify User Group . . . . . 547
Get User Groups . . . . . 548
Get User Group . . . . . 548
Get User Group Details . . . . . 549
Activate User Group . . . . . 550
Validate Login . . . . . 550
Change Password . . . . . 551
Set Recovery Question . . . . . 551
Get Recovery Question . . . . . 551
Add Contact . . . . . 551
Add Contact Info . . . . . 552
Modify Contact . . . . . 553
Get Contact . . . . . 553
Delete Contact . . . . . 554
Input Parameters Description . . . . . 554
X-Core APIs . . . . . 556
Add Code Master Value . . . . . 556
Modify Code Master Value . . . . . 556
Set Code Master Value Status . . . . . 557
Get Code Master Value . . . . . 557
Get Code Master Description . . . . . 557
Add Code Master Type . . . . . 558
Modify Code Master Type . . . . . 558
Raise Error . . . . . 559
Raise Exception . . . . . 559
Get Document Name . . . . . 560
Get Links . . . . . 560
Collate Elements . . . . . 561
Ignore Equals Case . . . . . 561
Get All Documents . . . . . 562
Get Document Properties . . . . . 562
Get Folders . . . . . 563
Get Supported Data Types . . . . . 564
Get Tables . . . . . 564
XML Sort . . . . . 565

Input Parameters Description . . . . . 566
MDM Meta Service APIs . . . . . 567
Start Timer for Publication . . . . . 567
Get Publication Objects . . . . . 567
Load ThesaurusDef . . . . . 567
Load Business Rule Definition . . . . . 567
Delete Business Rule Definition . . . . . 567
Update Business Rule Definition . . . . . 568
Refresh Business Rules . . . . . 568
Create Global Temp Tables . . . . . 568
Create Auth Views . . . . . 568
Get Service Names . . . . . 569
Get Document Names . . . . . 569
Get Property Names . . . . . 569

**Appendix L: Non MDM APIs** . . . . . 570

Non MDM APIs . . . . . 570
Get Non MDM Document Names . . . . . 570
Get Non MDM Document Properties . . . . . 570
Get Properties of selected Non MDM Documents . . . . . 571
Reload Non MDM Documents . . . . . 572
Is Non MDM Documents Registered . . . . . 572

**Appendix M: DB Based Locking Utility** . . . . . 573

Introduction . . . . . 573
Concept . . . . . 573
Setup . . . . . 574
Change xserver.xml . . . . . 574
Define a lock specification file . . . . . 574
Change service config file . . . . . 575
API specification . . . . . 575

**Appendix N: Export Framework** . . . . . 577

Introduction . . . . . 577
Export Functionality . . . . . 577

Features of Export Framework . . . . . 577
Key features of the export framework: . . . . . 578
Steps involved in using export framework: . . . . . 578
Defining export definition . . . . . 578
Define export definition . . . . . 579
Define delta export filter . . . . . 579
Define fields to be exported . . . . . 580
Define fields which would be used in export filter . . . . . 581
Loading Export Definition . . . . . 581
Creating export/staging table schema . . . . . 582
Invoking/Triggering export . . . . . 582
Export Framework Schema . . . . . 584
Potential enhancements . . . . . 585
Appendix 1 . . . . . 585

**Appendix O: Purge Framework** . . . . . 590

Introduction . . . . . 590
Purge Functionality . . . . . 590
Features of Purge Framework . . . . . 591
Key Features of the Export Framework . . . . . 591
Steps involved in using purge framework: . . . . . 591
Defining Purge Definition . . . . . 591
Assumptions . . . . . 594
Loading Purge Definition . . . . . 595
Creating Purge Backup/Shadow Table Schema . . . . . 595
Invoking/Triggering Purge . . . . . 595
Invoke purge command . . . . . 595
Purge Framework Schema . . . . . 596
Potential Enhancements . . . . . 597
Appendix 1 . . . . . 597

**Appendix P: Search Framework** . . . . . 599

Introduction . . . . . 599
Search Functionality . . . . . 599
Search Definition . . . . . 601
Selected properties . . . . . 603

Fetch document links . . . 603
Normal filters . . . 603
Domain filters . . . 604
Rule filters . . . 609
Specifying additional search criteria . . . 613
Interfaces to Search Framework . . . 614
Loading the Search Definition . . . 618
Appendix 1 . . . 619

**Appendix Q: State Machine** . . . 622

Introduction . . . 622
State Model . . . 622
State Model specification . . . 623
Mapping Condition Specification and Event Mapping . . . 625
State Model Association . . . 626
Exception States . . . 627
State Document . . . 628
Interfaces to State Machine . . . 628
RAISE_STATE_EVENT . . . 629
PREVIOUS_STATE_EVENT . . . 630
Obtaining event name for a state in a state model . . . 630
Loading State Model Files . . . 631
Parallel States . . . 631
Example of a State Model Spec . . . 633
FAQ . . . 637

**Appendix R: Workflow Specification Framework** . . . 639

Introduction . . . 639
Typical Workflow Example . . . 639
Typical Node Example . . . 640
Transaction Scope in Workflow . . . 640
Event Descriptors . . . 641
Built-In Workflow Nodes . . . 642
Start Node . . . 642
Task Node . . . 644
Decision Node . . . 644

Branch Node . . . . . 645
Event Node . . . . . 646
And Node . . . . . 647
Or Node . . . . . 647
Timer Node . . . . . 647
Sub-Workflow Node . . . . . 648
Done Node . . . . . 650

**Appendix S: Messaging and E-mail Service** . . . . . 652

Introduction . . . . . 652
Messaging Functionality . . . . . 652
Message Events . . . . . 654
Event Definition . . . . . 654
Message Template . . . . . 655
Recipient Groups . . . . . 660
RAISE MESSAGE EVENT . . . . . 664
E-mail Service . . . . . 664
Usage . . . . . 665

**Appendix T: Test Service** . . . . . 667

Introduction . . . . . 667
Goals . . . . . 667
Test service setup . . . . . 668
Configuration file . . . . . 668
Batch files . . . . . 669
Other files . . . . . 669
Test files . . . . . 669
Dictionary of Xpath functions and Xcore actions specific to Test service . . . . . 680
Steps to run PSR tests . . . . . 686
FAQ . . . . . 686

**Appendix U: MDM_SYS_ENT_STATE** . . . . . 688

Introduction . . . . . 688
SYS_ENT_STATE Attribute . . . . . 688
MDM SYS_ENT_STATE Business Use Case . . . . . 690

**Appendix V: Multiple Table Updates using DocView** . . . . . 692

Introduction . . . . . 692
Multiple Tables from DocView . . . . . 692

**Appendix W: Parent-Child Validation** . . . . . 696

Introduction . . . . . 696
Parent-Child Validation . . . . . 696

**Appendix X: Dashboard Technology Implementation** . . . . 701

Introduction . . . . . 701
Implementation Steps . . . . . 701
MDM Plugin Features . . . . . 702
Sample PGL Files . . . . . 705

**Appendix Y: Angular UI for Custom Workflows** . . . . . 710

Introduction . . . . . 710
Implementation Process . . . . . 711
Get a Sample Project and Check for Various Configuration . . . . . 711
Add Custom Services into Sample Project . . . . . 717
Production Build of Custom Project . . . . . 719
Deploy Custom Application in MDM . . . . . 719
Create Left Navigator Link . . . . . 720
Access Custom Application from Left Navigator . . . . . 721

**Appendix Z: Error Handling for Unavailable Shared Services** 722

Introduction . . . . . 722
Implementation Steps . . . . . 722

**Appendix AA: Frequently Asked Questions** . . . . . 724

FAQs . . . . . 724

# List of Figures

Figure 1: PGL Framework Architecture. . . . . 6
Figure 2: PGL Framework Flow Diagram . . . . . 7
Figure 3: Hello World Example . . . . . 10
Figure 4: Simple Container. . . . . 30
Figure 5: Tabbed Container . . . . . 31
Figure 6: Simple Step . . . . . 33
Figure 7: Tabbed Step. . . . . 34
Figure 8: Steps . . . . . 34
Figure 9: Search frame before the results. . . . . 37
Figure 10: Search frame with results . . . . . 39
Figure 11: Search Container to populate a table. . . . . 39
Figure 12: Text Area. . . . . 42
Figure 13: Text Area Example . . . . . 43
Figure 14: Button . . . . . 47
Figure 15: Define condition expression using userAction . . . . . 47
Figure 16: Attach condition to connector. . . . . 48
Figure 17: Button Group. . . . . 50
Figure 18: Check Box. . . . . 52
Figure 19: Check Box Example . . . . . 53
Figure 20: Radio Button . . . . . 56
Figure 21: Radio Button Example . . . . . 58
Figure 22: Display Field. . . . . 61
Figure 23: Display Field Example . . . . . 63
Figure 24: Chart Field. . . . . 66
Figure 25: Entry Field. . . . . 68
Figure 26: Entry Field Example . . . . . 69
Figure 27: Placeholder Attribute Example . . . . . 73
Figure 28: Datalist Example . . . . . 74
Figure 29: Field Group . . . . . 78
Figure 30: Field Group . . . . . 79
Figure 31: Field Group . . . . . 80
Figure 32: Field Space . . . . . 82

Figure 33: Divider. . . . . 83
Figure 34: Drop Down . . . . . 85
Figure 35: Option Example . . . . . 88
Figure 36: Option . . . . . 91
Figure 37: Horizontal Rule . . . . . 93
Figure 38: Header . . . . . 94
Figure 39: Help . . . . . 95
Figure 40: Validation Message . . . . . 97
Figure 41: Date Range . . . . . 100
Figure 42: Page Group . . . . . 104
Figure 43: Chart . . . . . 106
Figure 44: Login . . . . . 110
Figure 45: Navigation . . . . . 112
Figure 46: Pad. . . . . 115
Figure 47: Pad Item . . . . . 117
Figure 48: Image. . . . . 119
Figure 49: Image Example . . . . . 120
Figure 50: Image Link Field . . . . . 122
Figure 51: Link . . . . . 125
Figure 52: Link Field . . . . . 127
Figure 53: Link . . . . . 127
Figure 54: URL. . . . . 131
Figure 55: UI:DIV . . . . . 133
Figure 56: Simple Table . . . . . 135
Figure 57: Inverted Table . . . . . 138
Figure 58: Tree Cell Example. . . . . 139
Figure 59: Inverted Table . . . . . 140
Figure 60: Table Example . . . . . 140
Figure 61: Columns Frozen in the Table . . . . . 141
Figure 62: Grid . . . . . 152
Figure 63: Choose Field . . . . . 158
Figure 64: Choose Field Example . . . . . 159
Figure 65: Where . . . . . 162
Figure 66: Dynamic PGL Example . . . . . 169
Figure 67: Excel Pivot . . . . . 172
Figure 68: Horizontal Tree Example . . . . . 175

Figure 69: Pivot Example . . . . . 179
Figure 70: Vertical Tree Example . . . . . 190
Figure 71: Work Book Example . . . . . 192
Figure 72: WorkBook Example . . . . . 197
Figure 73: WorkBook Example . . . . . 198
Figure 74: UI:DIV . . . . . 201
Figure 75: DO_DB_Persist Framework Architecture . . . . . 372
Figure 76: DO_DB_Persist Process Flow . . . . . 375
Figure 77: Sample UI for Angular Material Icons on Toolbar . . . . . 437
Figure 78: Sample UI for Image Menu on Toolbar . . . . . 438
Figure 79: Sample UI for Button and Button Menu on Toolbar . . . . . 438
Figure 80: Toggle Filter . . . . . 439
Figure 81: Inline Editing . . . . . 440
Figure 82: Inline Menu Option . . . . . 441
Figure 83: Header Icons (Customize, Multi-Column Sort, Export) . . . . . 441
Figure 84: Sample UI for Card View . . . . . 442
Figure 85: Sample UI for Card View . . . . . 443
Figure 86: MDM Sample Form UI . . . . . 444
Figure 87: Sample UI for Button Display under each Tab . . . . . 445
Figure 88: PGL Examples main . . . . . 449
Figure 89: Basic Controls . . . . . 450
Figure 90: Advanced Controls . . . . . 450
Figure 91: Samples . . . . . 451
Figure 92: Properties Editor . . . . . 452
Figure 93: Search . . . . . 453
Figure 94: Search . . . . . 454
Figure 95: Search . . . . . 454
Figure 96: Search Complete . . . . . 455
Figure 97: Directory Structure . . . . . 501
Figure 98: Solution Setup . . . . . 503
Figure 99: Select Services . . . . . 503
Figure 100: Generate Schema . . . . . 503
Figure 101: Setup Static Data . . . . . 504
Figure 102: Operation/Rule . . . . . 504
Figure 103: Run Rule . . . . . 505
Figure 104: Search Framework . . . . . 600

Figure 105: Sub Workflow Node . . . . . 650
Figure 106: Interactions between Messaging service and other services . . . . . 653
Figure 107: State Transition Model . . . . . 689
Figure 108: State Transition Model . . . . . 691
Figure 109: Create View. . . . . 693
Figure 110: Create Data Persist File . . . . . 694
Figure 111: Create Rule . . . . . 695
Figure 112: Sample PGL Chart Files . . . . . 706
Figure 113: Sample Pie_Chart . . . . . 706
Figure 114: Sample Pie_Bar_Chart . . . . . 707
Figure 115: Sample Pie_Line_Bar_Chart. . . . . 708
Figure 116: Sample Pie_Line_Bar_Gauge_Chart . . . . . 709
Figure 117: Angular UI Architecture . . . . . 710

# List of Tables

Table 1: Required Attributes . . . . . 32
Table 2: Optional Attributes . . . . . 32
Table 3: Required Attributes . . . . . 35
Table 4: Optional Attributes . . . . . 35
Table 5: Optional Attributes . . . . . 40
Table 6: Required Attributes . . . . . 43
Table 7: Optional Attributes . . . . . 44
Table 8: Required Attributes . . . . . 48
Table 9: Optional Attributes . . . . . 49
Table 10: Required Attributes . . . . . 50
Table 11: Optional Attributes . . . . . 51
Table 12: Required Attributes . . . . . 53
Table 13: Optional Attributes . . . . . 53
Table 14: Required Attributes . . . . . 58
Table 15: Optional Attributes . . . . . 58
Table 16: Required Attributes . . . . . 63
Table 17: Optional Attributes . . . . . 63
Table 18: Optional Attributes . . . . . 67
Table 19: Required Attributes . . . . . 70
Table 20: Optional Attributes . . . . . 70
Table 21: Required Attributes . . . . . 74
Table 22: Optional Attributes . . . . . 74
Table 23: Optional Attributes . . . . . 76
Table 24: Optional Attributes . . . . . 81
Table 25: Optional Attributes . . . . . 83
Table 26: Required Attributes . . . . . 84
Table 27: Optional Attributes . . . . . 84
Table 28: Required Attributes . . . . . 88
Table 29: Optional Attributes . . . . . 88
Table 30: Required Attributes . . . . . 91
Table 31: Optional Attributes . . . . . 92
Table 32: Optional Attributes . . . . . 96

Table 33: Optional Attributes . . . . . 99
Table 34: Required Attributes . . . . . 100
Table 35: Optional Attributes . . . . . 100
Table 36: Optional Attributes . . . . . 102
Table 37: Optional Attributes . . . . . 104
Table 38: Optional Attributes . . . . . 107
Table 39: Optional Attributes . . . . . 108
Table 40: Optional Attributes . . . . . 109
Table 41: Optional Attributes . . . . . 110
Table 42: Optional Attributes . . . . . 113
Table 43: Optional Attributes . . . . . 113
Table 44: Optional Attributes . . . . . 116
Table 45: Optional Attributes . . . . . 118
Table 46: Optional Attributes . . . . . 119
Table 47: Optional Attributes . . . . . 120
Table 48: Required Attributes . . . . . 122
Table 49: Optional Attributes . . . . . 122
Table 50: Optional Attributes . . . . . 125
Table 51: Required Attributes . . . . . 128
Table 52: Optional Attributes . . . . . 128
Table 53: Optional Attributes . . . . . 131
Table 54: Required Attributes . . . . . 133
Table 55: Optional Attributes . . . . . 133
Table 56: Required Attributes . . . . . 143
Table 57: Optional Attributes . . . . . 144
Table 58: Required Attributes . . . . . 146
Table 59: Optional Attributes . . . . . 146
Table 60: Required Attributes . . . . . 148
Table 61: Optional Attributes . . . . . 149
Table 62: Optional Attributes . . . . . 152
Table 63: Optional Attributes . . . . . 154
Table 64: Optional Attributes . . . . . 156
Table 65: Required Attributes . . . . . 159
Table 66: Optional Attributes . . . . . 159
Table 67: Required Attributes . . . . . 162
Table 68: Optional Attributes . . . . . 162

Table 69: Required Attributes . . . . . 165
Table 70: Optional Attributes . . . . . 165
Table 71: Required Attributes . . . . . 169
Table 72: Optional Attributes . . . . . 172
Table 73: Optional Attributes . . . . . 175
Table 74: Optional Attributes . . . . . 176
Table 75: Optional Attributes . . . . . 179
Table 76: Optional Attributes . . . . . 181
Table 77: Optional Attributes . . . . . 182
Table 78: Optional Attributes . . . . . 183
Table 79: Required Attributes . . . . . 184
Table 80: Optional Attributes . . . . . 184
Table 81: Optional Attributes . . . . . 187
Table 82: Optional Attributes . . . . . 188
Table 83: Optional Attributes . . . . . 190
Table 84: Optional Attributes . . . . . 198
Table 85: Optional Attributes . . . . . 199
Table 86: Required Attributes . . . . . 201
Table 87: Optional Attributes . . . . . 201
Table 88: Pattern letters . . . . . 412
Table 89: Time Zone Pattern . . . . . 414
Table 90: Responses for POST API . . . . . 425
Table 91: Responses for GET API Request . . . . . 427
Table 92: Responses for PUT API . . . . . 429
Table 93: Responses for Delete API. . . . . 430
Table 94: XML and Equivalent JSON Request . . . . . 431
Table 95: X-Rule (XML) and Equivalent JSON Response . . . . . 433
Table 96: Basic Controls . . . . . 447
Table 97: Advanced Controls . . . . . 448
Table 98: Samples. . . . . 448
Table 99: Files in the ruleLangExamples Directory . . . . . 502
Table 100: Lock_Table. . . . . 573
Table 101: PURGE_LOG Table. . . . . 596
Table 102: PURGE_LOG_LN Table . . . . . 596
Table 103: SendMail API . . . . . 665

# SECTION A UI Reference

Section A provides link to the various chapters on UI Reference.


- Chapter 1: “Page Layout.”
- Chapter 2: “Getting Started.”
- Chapter 3: “New HTML5 Compliant UI.”
- Chapter 4: “Basic Controls.”
- Chapter 5: “Advanced Controls.”
- “Appendix A  Using the PGL Example Project”
- “Appendix B  Hierarchy Viewer-Custom Workflow”
- “Appendix Y  Angular UI for Custom Workflows”
- “Appendix Z  PGL UI Framework”

# CHAPTER 1 Page Layout

### What's In This Chapter

This chapter gives you an introduction to Page layout (PGL) and an overview of the PGL specification.

Topics include:


- Introduction
- Page Layout
- Getting to Know PGL
- Document Structure
- Element Realization
- Element Attributes
- PGL Framework Architecture
- PGL Generic Notes


## Introduction

Any workflow typically consists of system-to-system interactions as well as system-to-human interactions. The system-to-human interactions are described through the use of user interfaces. The Teradata Studio provides the tools needed to generate User Interfaces (UI) without having to do any low level coding. All the typical notions of labels, text boxes, tables and other UI components are available as controls within Teradata Studio and can be used to build a user interface. The user interface is described in a language called Page Layout (PGL).

## Page Layout

Page Layout is a powerful and flexible markup language to describe the user interfaces of the dynamic web applications. PGL uses declarative language to describe the components and the layout of the "page". In a Model-View-Controller architecture, PGL represents the view aspect with no controller logic or defined data model.

PGL can be authored with a text editor such as the Notepad for windows or using any standard XML editor. The MDM application framework can manifest or reconstruct objects defined in the mark-up using the UI Services of MDM framework.

# Getting to Know PGL

The user interface descriptions written in any programming language are frozen at compile time, irrespective of the tools you use. For example, to change the shading of an icon from one style to another, it requires re-compilation of the existing application. Dynamic reconstruction of the objects using mark-up introduces flexibility and allows personalization of the user interface without the complexities of a programming language.

The choice of XML coupled with the power of XSL transformations opens up numerous possibilities as well. The PGL mark-up makes it easy to mix and match loosely coupled components to create intricate designs that would be difficult and relatively expensive when developed using other methodologies.

PGL conforms to the W3C recommendation for XML Version 1.0. Use of the W3C standard allows other standards such as XSL Transformations to manipulate or customize the view. PGL can also be easily parsed, manipulated using standard XML tools.

PGL defines a rich collection of UI element types such as the following:

- **Container**. Represents a frame.
- **Step**. Represents a panel.
- **Search**. Represents a *Search UI framework*.
- **Table**. Represents a *tabular data view* with paging support and so on.

The process of object realization or reconstruction causes an element to manifest itself at run-time in a form dictated by the class that implements the element's core functionality. The look-and-feel of the elements conforms with the *Teradata MDM UI guidelines* (frog).

## Object Association

The following are the various types of association available.

- Association by Element Containment

  The meaning of an association described by containment depends on the types of both the parent element and the child element. During run-time, a given element may assimilate, different types of child elements. For describing view-hierarchies the common type of object association is by Element Containment.
- Association by URI

  You can use URI association to specify the location of image resource or help document or any "reference-able static document.
- Association by URL

  All object associations that can use an URI can potentially use an URL to reference an external documentation.Thus keeping the content returned by server as thin as possible. For example, returning a help documentation might unnecessarily load the server and the client. An URL association for such elements might be beneficial.

### View Hierarchies

It is easy to describe complex view hierarchies by embedding one or more presentation components inside a step or by embedding one or more steps inside a container.

# Document Structure

PGL documents must meet the requirements of well-formed XML documents. A PGL document must start with a mandatory xml declaration identifying at least the version of the XML specification that the document conforms to. An optional encoding parameter identifies the encoding scheme used in the document. By default the UTF-8 is assumed.

The XML declaration should be followed by the Page-Group element which is the root-element for a PGL document. You can either use a document-wide default namespace by using the xmlns attribute of the Page-Group element or by using the namespace as a prefix to the PGL-UI elements.

# Element Realization

The PGL-UI element defines the default mapping between an element type and a class used to realize the element of that type.

# Element Attributes

PGL has a rich library of UI elements with a set of attributes to capture the dynamic and static data, and customize the presentation of each of them.

Attributes for an element can be specified in any order without changing the behavior of the element.

```
<ui:display-field displayText=”Hello World!” id=”df1” name=”df1”/>
```

In the above example, the attributes—displayText, id, and name, can be in any order.

# PGL Framework Architecture

PGL programs are written in XML language, which makes the language to accommodate any new standards/frameworks without modification to existing programs. It is also possible to extend PGL language by adding any number of new features.

Main components of PGL architecture includes the following:

- PGL Loader

  It loads or caches all PGL files at application server startup time.

- PGL Workflow or Rule Execution Engine

  It executes XML commands of PGL workflow. This includes execution of nodes of workflow, rules and generating required input to UI template file that is., .pgl file. It gives input data to next component that is., PGL to HTML converter
- PGL to HTML converter

  It gets the input data from PGL Workflow or Rule execution engine and passes it to sub component HTML or JS generator. It includes the following two modules:

  - PGL compiler

    Compiles UI template file that is., .pgl file, to check template XML falls in line with the PGL specification given in ui.pgc file. It checks UI template XML has correct parent/child tags, attributes etc. and it throws exception if it violates the ui.pgc specification.
  - HTML/JS generator

    On successful compilation of .pgl file, HTML is generated by parsing the XML given in .pgl file and adding input data. Each PGL tag has its own HIJ (HTML In Java) handler to generate its HTML. HTML for all PGL tags is collected and passed to browser to paint on UI.

  The Figure 1 and Figure 2 displays the PGL framework architecture and PGL framework flow diagram.

Figure 1: PGL Framework Architecture

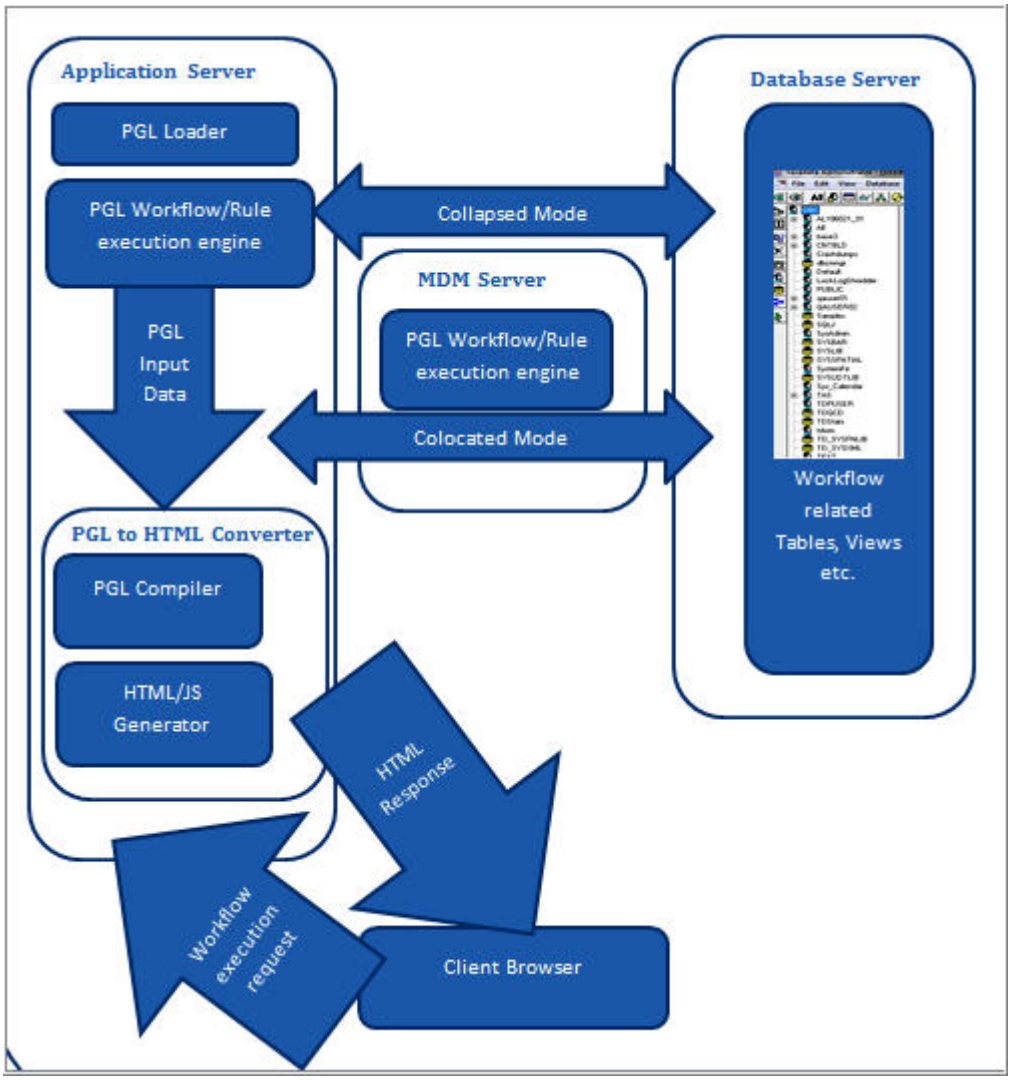
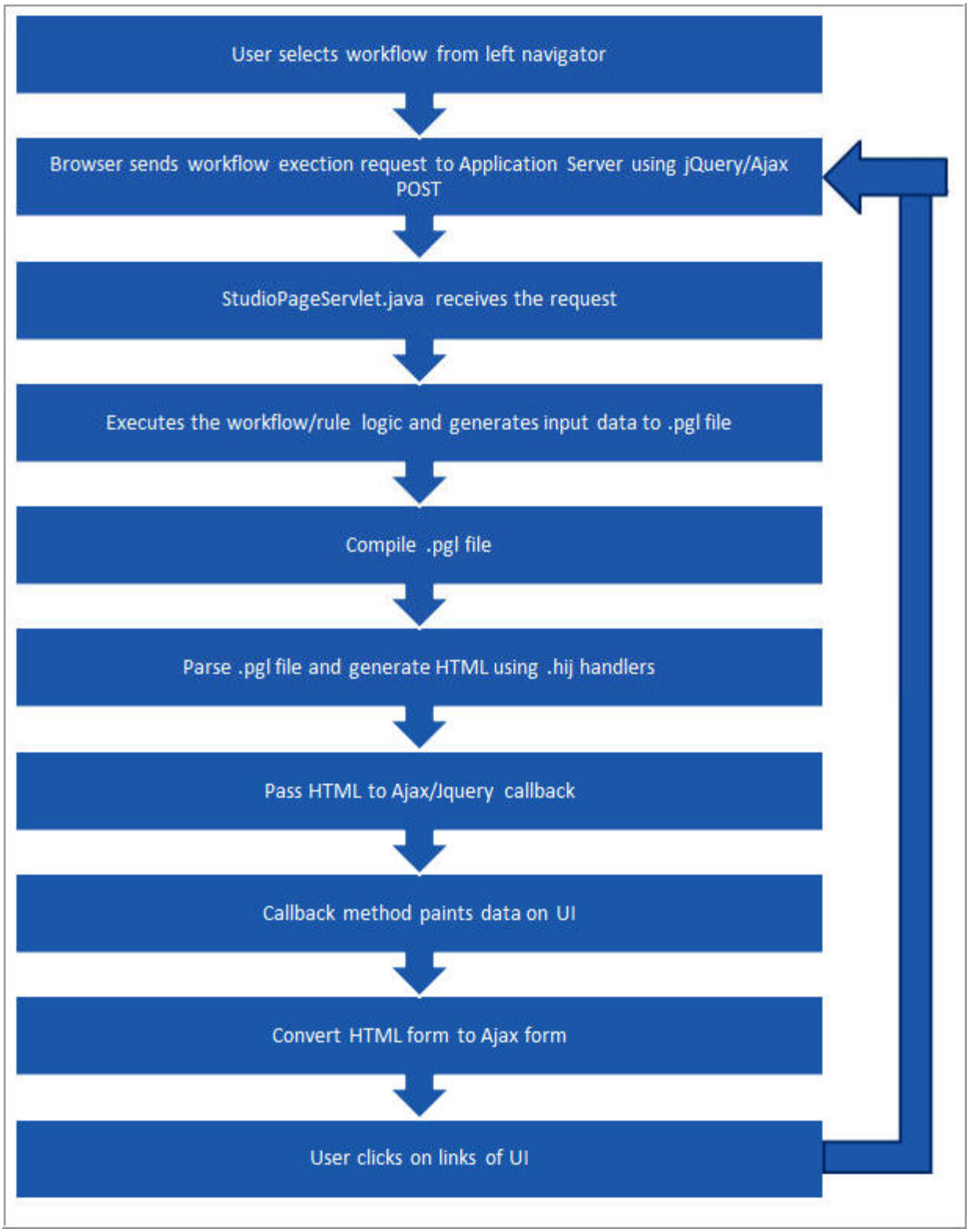


Figure 2: PGL Framework Flow Diagram



# PGL Generic Notes

This section provides the list of PGL generic notes:

1. For any PGL UI entry field description, it is recommended to enter both the **Id** and **name** fields as below:

   ```
   <ui:entry-field id="entryId" name="entryname"/>
   ```

2. The title attribute is added to PGL UI fields to display mouse over text for all PGL UI components. Also refer to *chapter UI Customization in MDM Platform Server Guide.*

3. A service parameter MDM_INSTALL_DIR in *<MDM_Install_Directory>\cfg\properties\xserver.xml* and *<MDM_Install_Directory>\cfg\properties\xserverweb.xml* holds the absolute MDM installation directory path. This parameter value can be retrieved and used from within the MDM PGL UI workflow or X-rules.

# CHAPTER 2 Getting Started

### What's In This Chapter

This chapter helps you get started with PGL.

Topics include:

- Introduction
- The Hello World Example

## Introduction

PGL documents (or XML documents that conform to PGL standards) processed by Teradata UI Services must conform to the Teradata schema. The document is constrained by elements that are supported in the schema. Although the schema may allow element types from other namespaces, the UI Services currently recognizes only the element types from the Teradata namespace.

## The Hello World Example

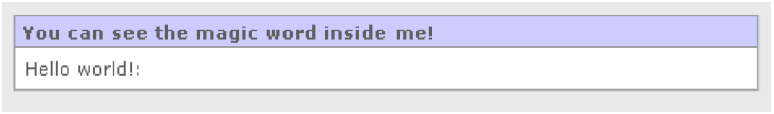
```
<?xml version="1.0" encoding="UTF-8">-----------------(1)
<ui:page-group xmlns:ui="http://www.teradata.com/ui">-------(2)
<ui:container>-------------------------------------(3)
<ui:step id="step_0" displayText="You can see the
  magic word inside me!">-----------------------(4)
<ui:field-group columns="1">-----------------(5)
   <ui:display-field displayText="Hello World!"
   id="df1" name="df1"/>---------------------(6)
</ui:field-group>
</ui:step>
</ui:container>
</ui:page-group>
```

In the above example:

1. It is the XML declaration.
2. Page Group. It is the root element of a PGL document from the Teradata MDM UI namespace. A page-group contains a page description.
3. Container. The steps of the Teradata MDM UI framework are contained in this.
4. Step. This is either a tab in a tabbed container or a step in a wizard container.
5. Field Group. It is the logical grouping of fields.
6. Display Field. It is the field component that displays its label and value in read-only mode.

The result of the above PGL in your web-browser is as below.

Figure 3: Hello World Example



# CHAPTER 3 New HTML5 Compliant UI

### What's In This Chapter

This chapter helps you get started with PGL.

Topics include:


- Introduction
- Major Framework Changes


# Introduction

MDM 3.04.01 release improved the look and feel of  the UIs by enhancing the PGL framework to generate HTML 5 compliant and compatible UIs . This PGL framework enhancement reduced lot of complexities in comparision to the earlier HTML generation process. It also makes it easy and simple for any future improvements in the framework.

This chapter describes the major changes done in PGL framework to make HTML compliant with HTML5/CSS3. It also describes the changes required in .pgl files to make them work in new UI. Custom PGLs upgraded to 3.4.1 release may require minimal changes to make the UI work properly. HTML 5 changes are supported on Internet Explorer10 and above, Firefox latest, and Chrome latest browsers.


- Major Framework Changes
- PGL Changes


# Major Framework Changes

The following framework modification changes the MDM look and feel without affecting the overall MDM functional compatibility.

## HTML/CSS Changes

HTML 5 compatibility changes affects the following items on MDM UI:

- Layout
- Header
- Left Navigation Menu
- Buttons
- Font/Color/Size/Padding

- Button Row
- Content Panels
- Breadcrumbs

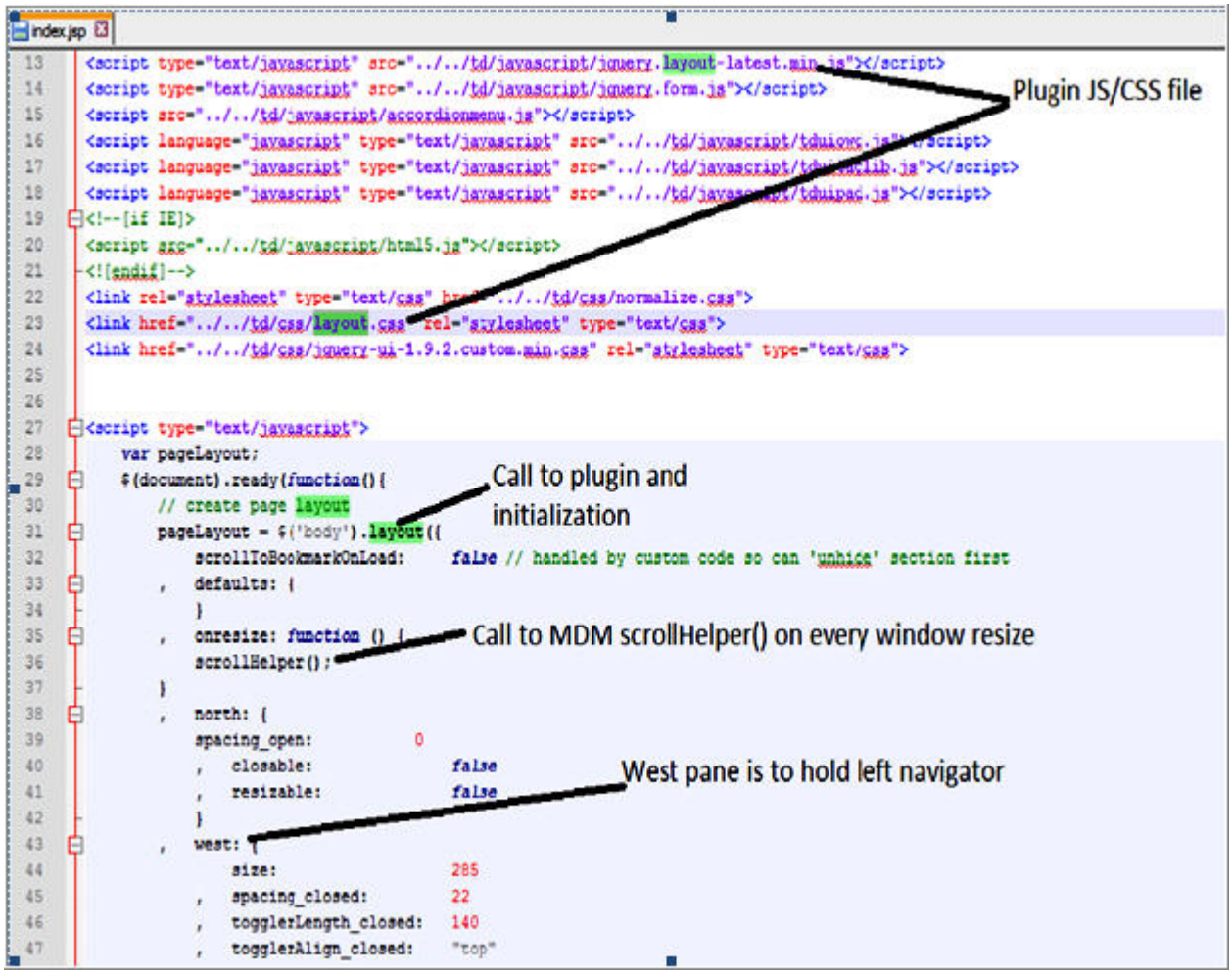
## Removal of Frames

Frames (<frame>/<frameset>) are not supported in HTML5. Below frames are removed and the same functionality is achieved with jQuery UI layout plugin:

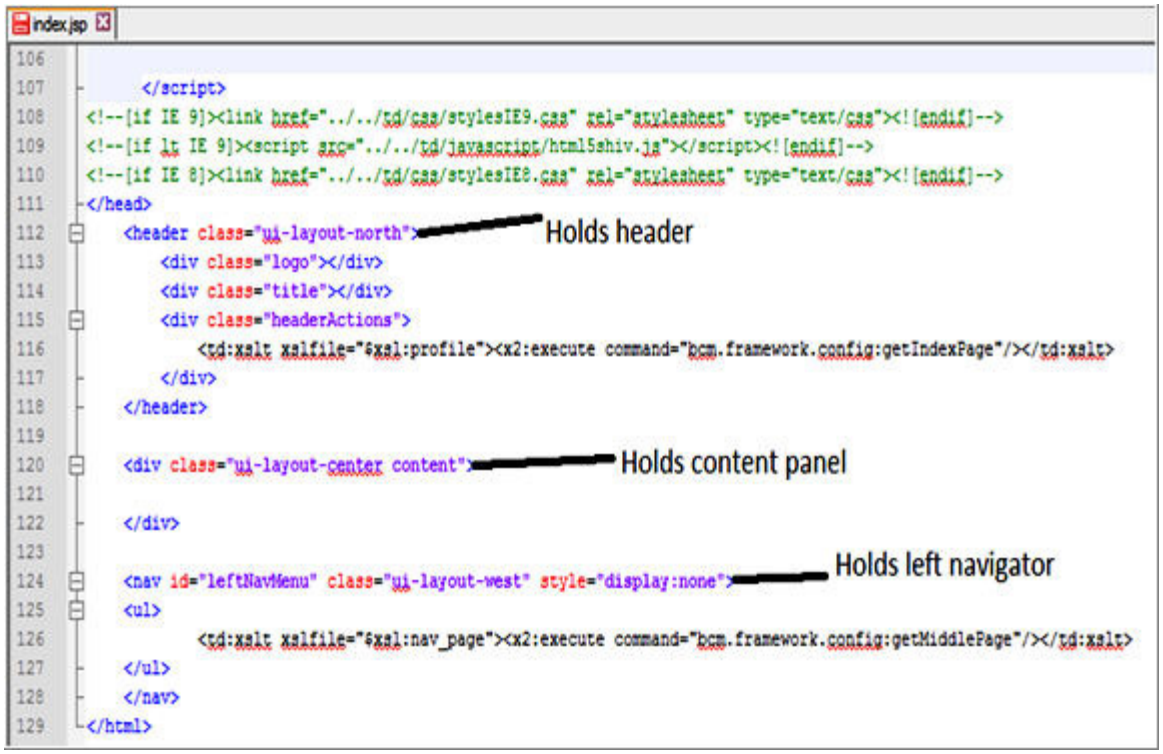
- navFrame (for holding left navigator)
- appFrame (for holding main content panel)
- tdui_shell_top (for holding header)
- tdui_shell_bottom (placeholder for excel download)
- tdui_shell_left (for border)
- tdui_shell_right (for border)

Information on new layout:

- Plugin: http://layout.jquery-dev.com/index.cfm
- Plugin Files: jquery.layout-latest.min.js, layout.css
- Used In: index.jsp

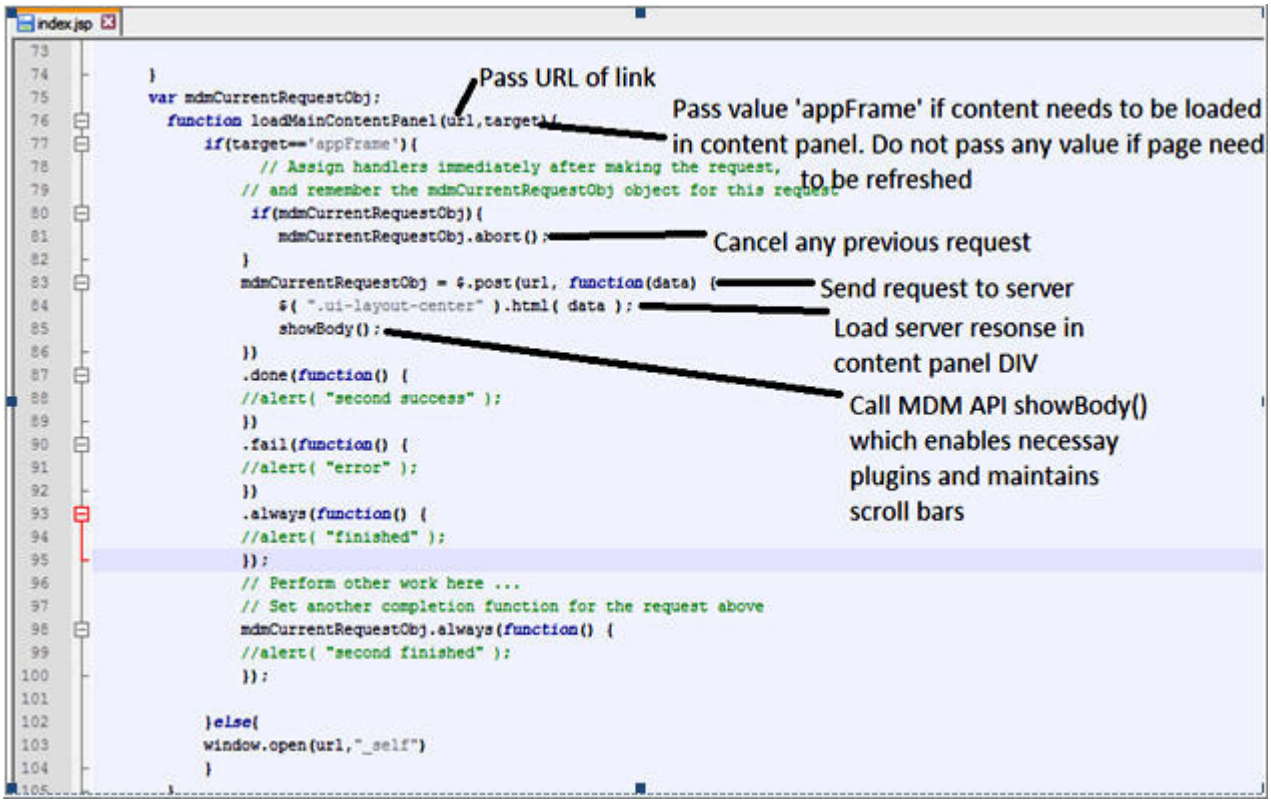



Below screenshot displays HTML used for layout.



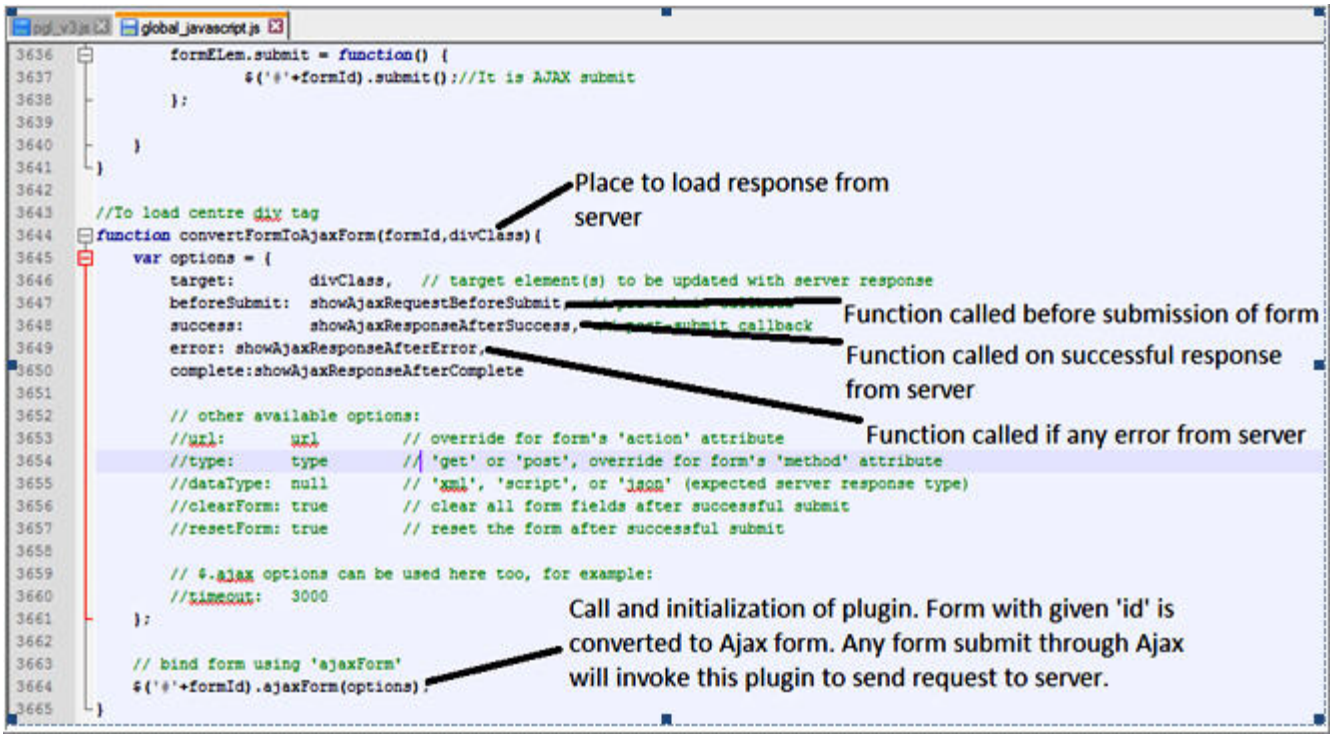

## Load Content Click of Links

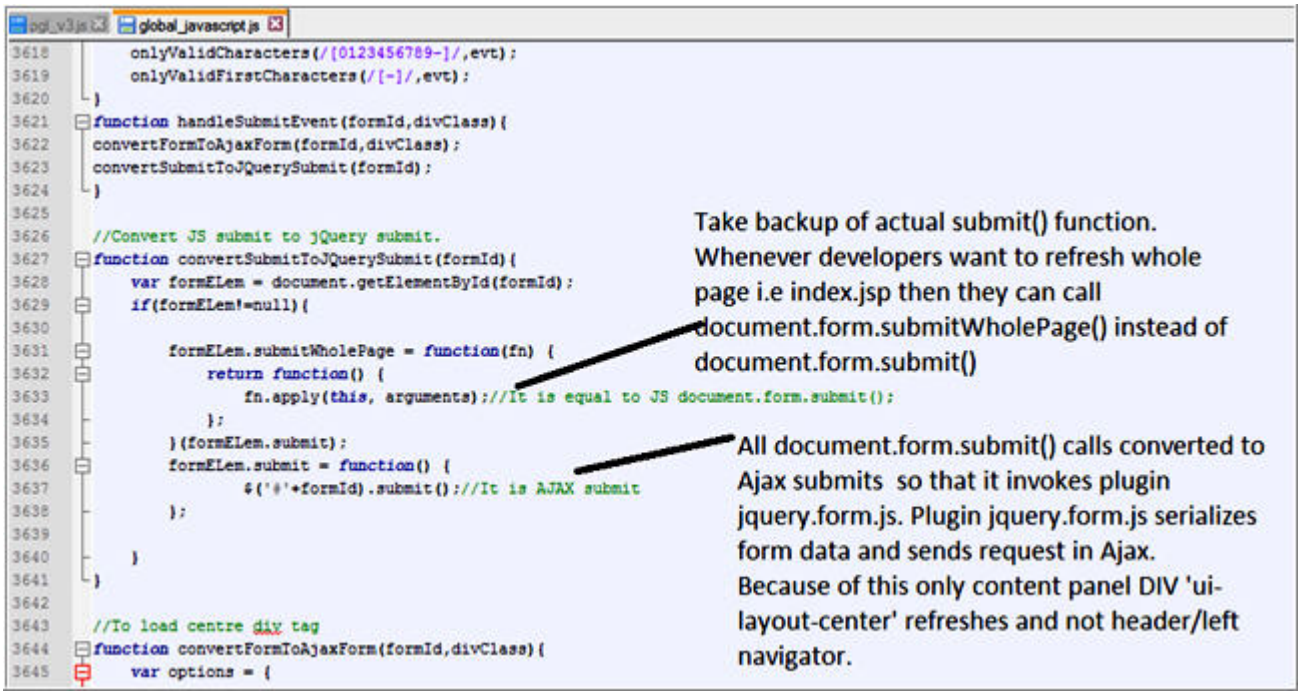
- Load of content on click of left navigator links or any links within MDM UI happens through MDM APIs loadMainContentPanel() and processLink().
- processLink() internally calls loadMainContentPanel()




## Form Submission (Load Content on Click of Buttons)

- PGL developers usually write JavaScript call document.form.submit() on click of any button.
- document.form.submit() refreshes whole page/frame and it worked well in old UI because page/UI is inside frame 'appFrame'. So it refreshes only frame 'appFrame' and not header/ left navigator
- New UI has no frames and document.form.submit() refreshes whole page that is index.jsp which includes header/left navigator. To avoid this JavaScript submit calls are converted to Ajax submits.
- Plugin: http://malsup.com/jquery/form/
- Plugin Files: jquery.form.js
- Used In: Index.jsp, global_javascript.js, pgl_v3.js




## File Downloads

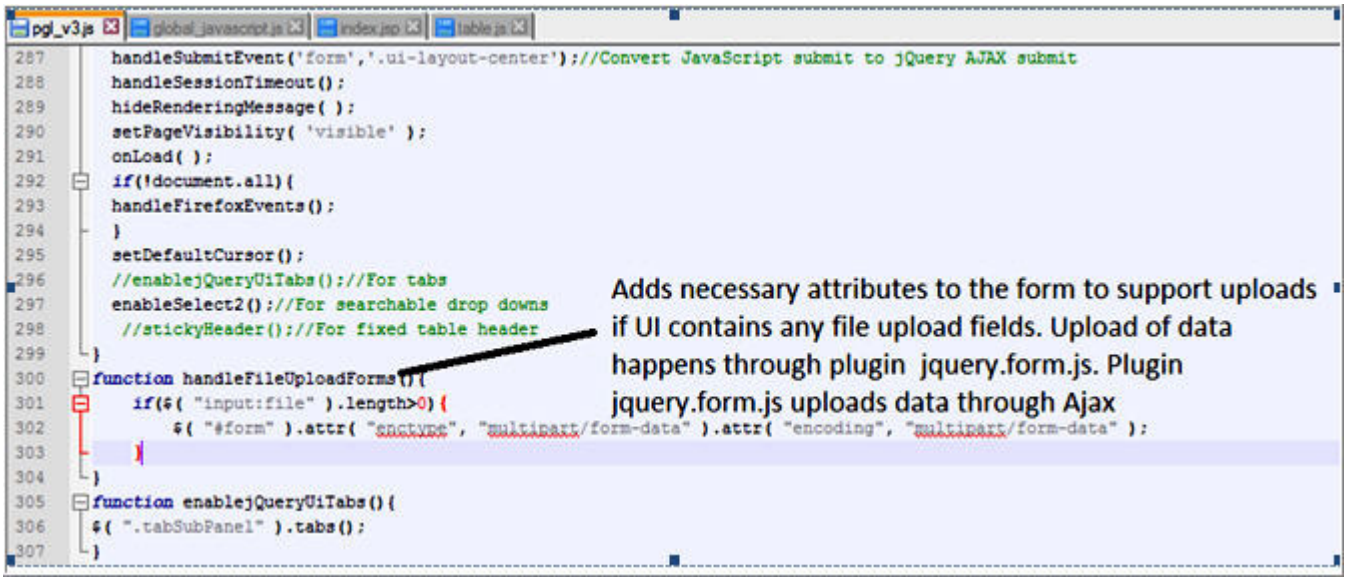
- File downloads need placeholder that is target at client side.
- Hidden frame 'tdui_shell_bottom' is used as target in old UI.
- Hidden iframe 'hiddenDownloader' is used as target in new UI.

```
function exportTablePGL(tableId,EXPORT_ALL)
{
  var pageId = document.form.PAGE_NAME.value;

  var prevAction = document.form.action;
  var prevTarget = document.form.target;

  if(document.form.MAX_ROWS)
  {
  var prevMaxRows = document.form.MAX_ROWS.value;
  var max = page_maxRowsExportable;
 //This is a pgl export.So Total no of exportable rows should be equals to no of total no of rows
  /*if (document.form.RECORD_COUNT.value > max)
  {
     core_alert("Your search Criteria resulted in {0} rows. Only {1} rows can be exported to excel. Please c
     return;
  }*/
  document.form.MAX_ROWS.value = document.form.RECORD_COUNT.value;
  }

  document.form.BUTTON_ID.value = "SYS_RELOAD";

  document.form.target = "hiddenDownloader";
  document.form.action=omxContextPath+ "/SearchExcelDownloadServlet?FILTER_ID="+ tableId+"&PAGE_ID="+pageId+"
  document.form.submitWholePage();

  document.form.action= prevAction;
  document.form.target= prevTarget;
```

'hiddenDownloader' is the id of hidden <iframe>.

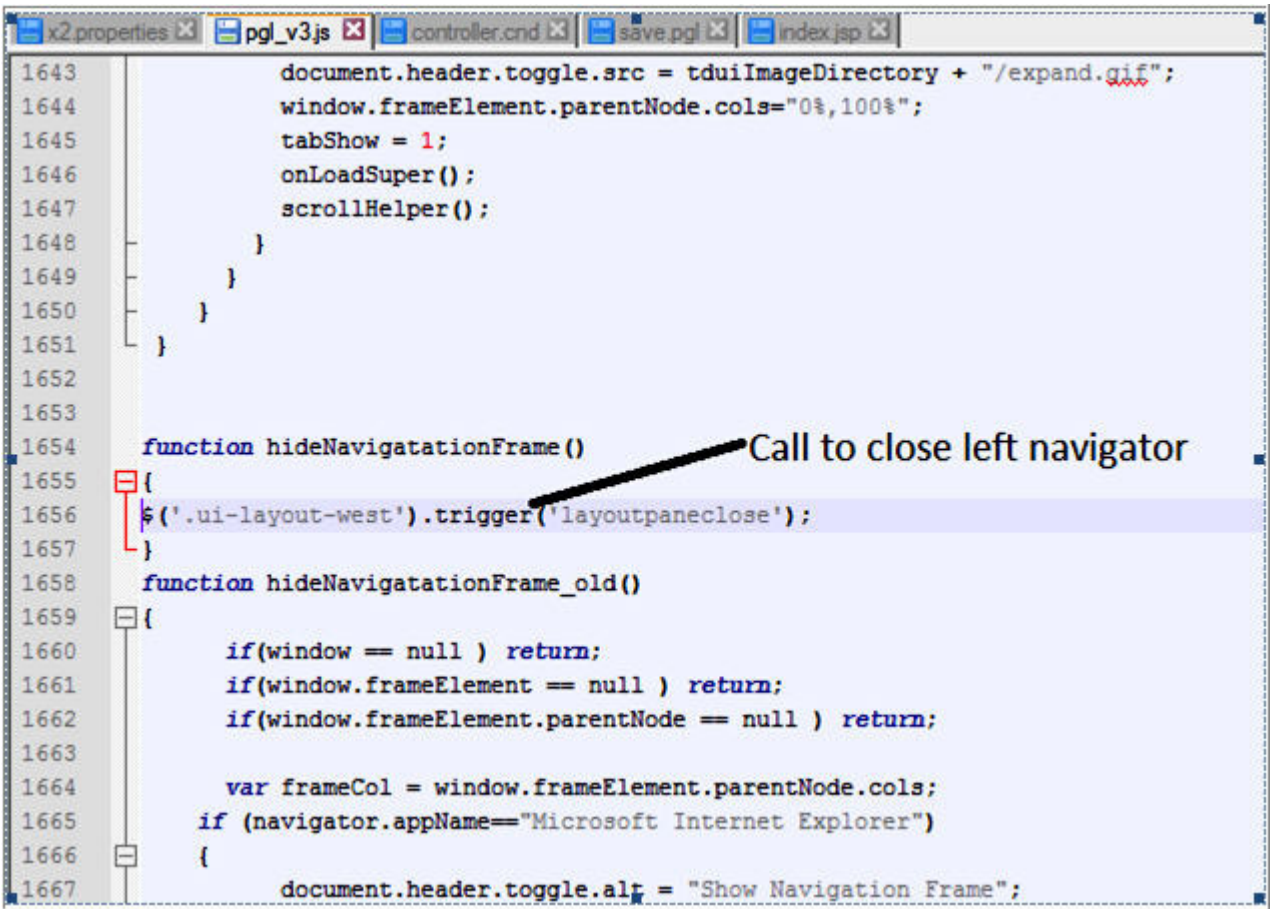
## File Uploads

- File upload happens through plugin jquery.form.js.
- Plugin jquery.form.js uploads data through Ajax
- Plugin: http://malsup.com/jquery/form/
- Plugin Files: jquery.form.js
- Used In: Index.jsp, global_javascript.js, pgl_v3.js




## Hide Left Navigator

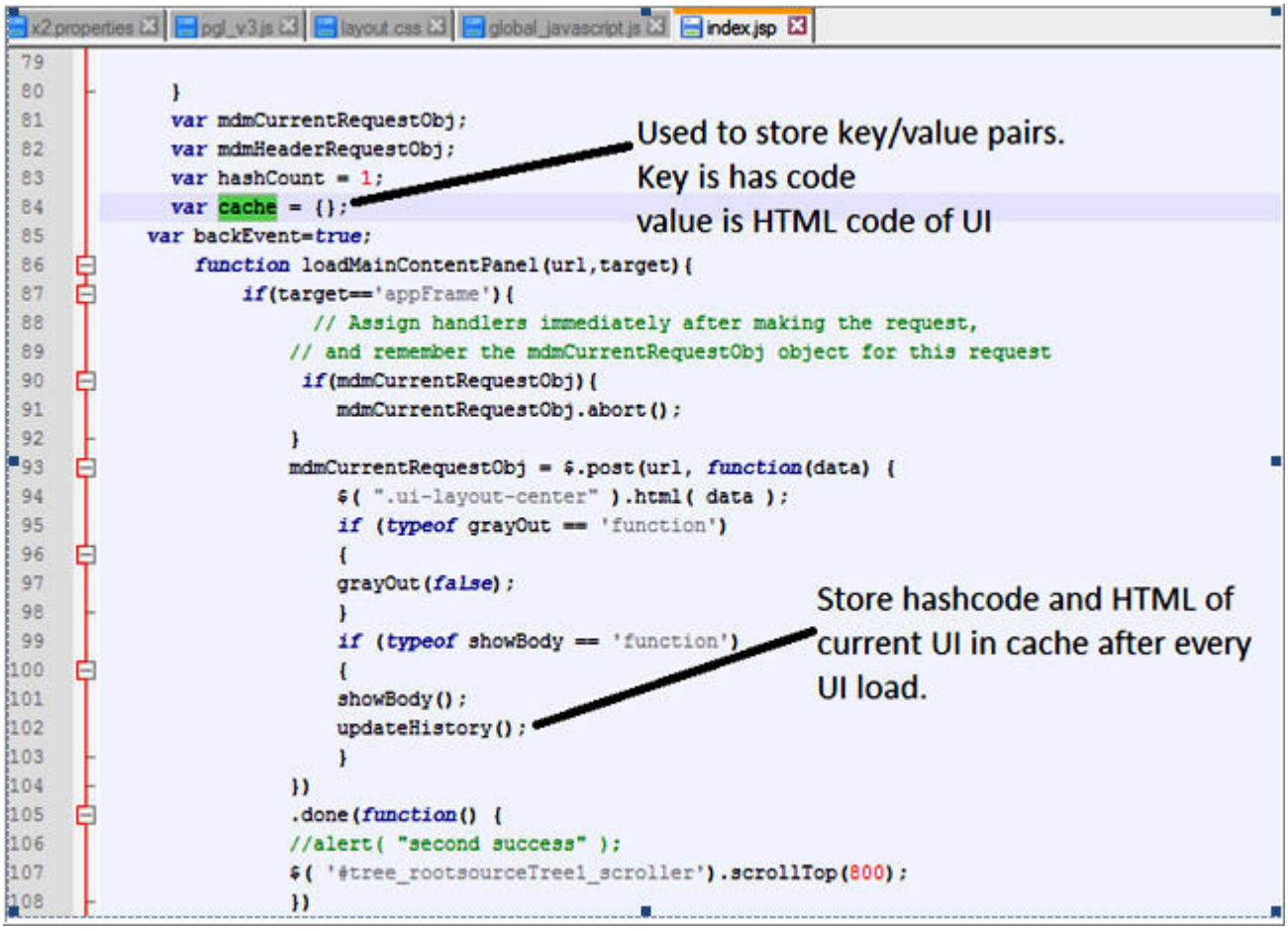
Some UIs require hiding left navigator on load of UI. Call API hideNavigatationFrame() to hide left navigator.



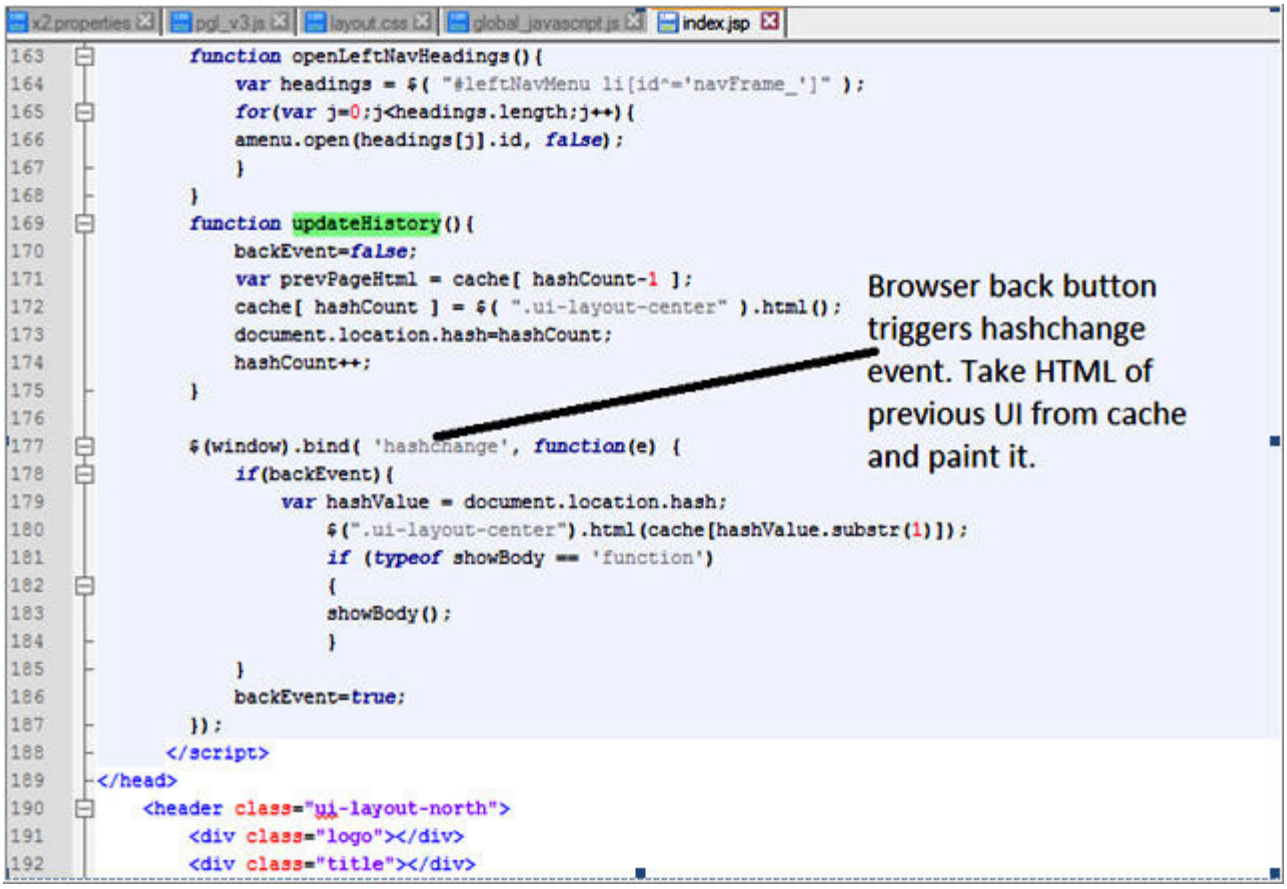
## History Maintenance

History of UIs maintained:

- To load previous UI on browser back button.
- To make command window.history.back() work as usual.
- To make command window.history.go() work as usual.




```
79
80      }
81      var mdmCurrentRequestObj;
82      var mdmHeaderRequestObj;
83      var hashCount = 1;
84      var cache = {};
85    var backEvent=true;
86        function loadMainContentPanel(url,target){
87            if(target=='appFrame'){
88                  // Assign handlers immediately after making the request,
89                // and remember the mdmCurrentRequestObj object for this request
90                 if(mdmCurrentRequestObj){
91                    mdmCurrentRequestObj.abort();
92                }
93                mdmCurrentRequestObj = $.post(url, function(data) {
94                    $( ".ui-layout-center" ).html( data );
95                    if (typeof grayOut == 'function')
96                    {
97                    grayOut(false);
98                    }
99                    if (typeof showBody == 'function')
100                   {
101                   showBody();
102                   updateHistory();
103                   }
104               })
105               .done(function() {
106               //alert( "second success" );
107               $( '#tree_rootsourceTree1_scroller').scrollTop(800);
108               })
```




## Minify JavaScript

Google closure compiler is used to minify JavaScript so that JavaScript loads faster. Minifying is done at build time and minified files are available at below location: *<MDM_Install_Directory>\web\mdmclient\td\javascript*

Source files are still available at: *<MDM_Install_Directory>\bre\ui\td\javascript*

Compiler used for JavaScript Minification: https://developers.google.com/closure/compiler/

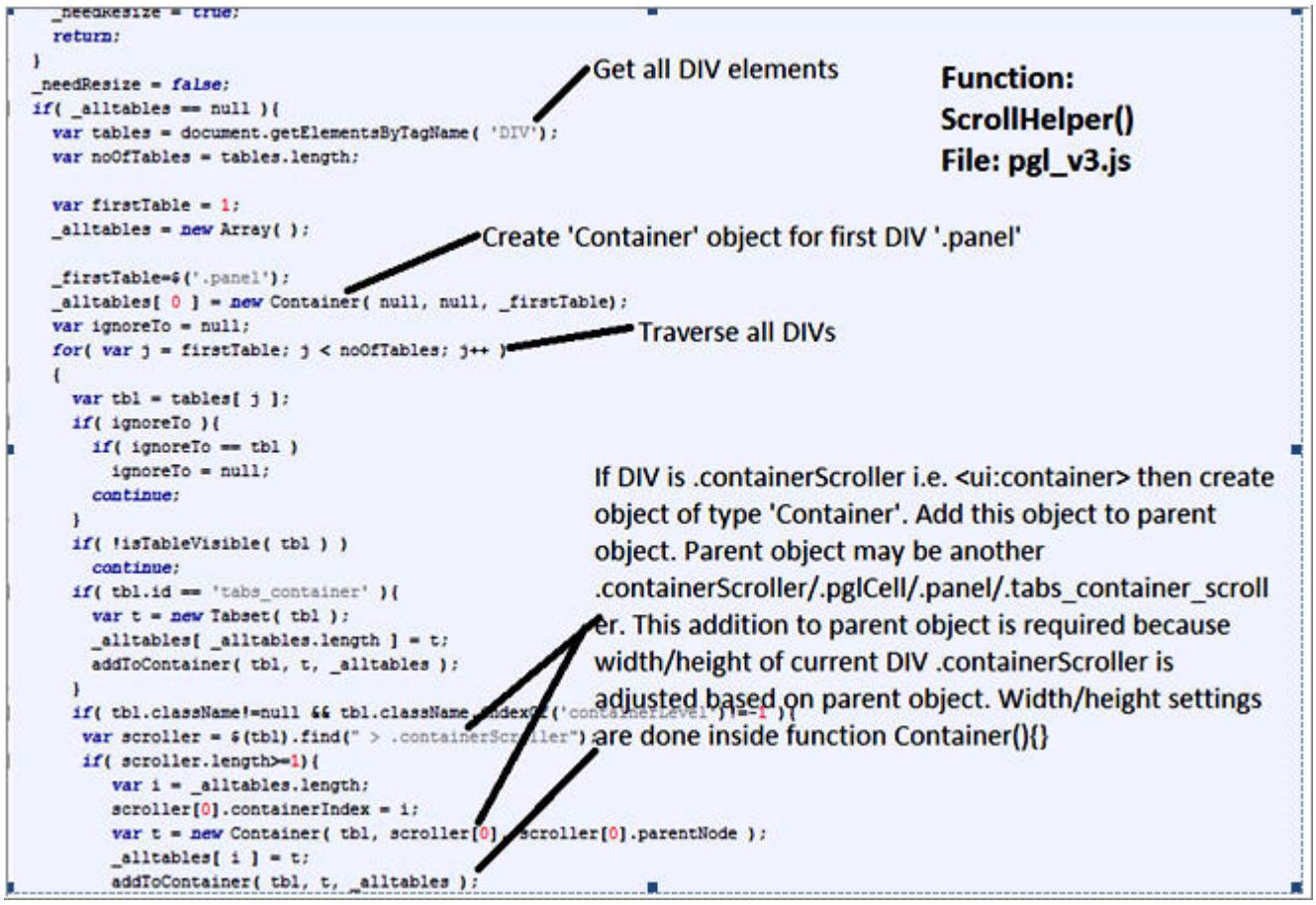
## Scroll Bar Maintenance

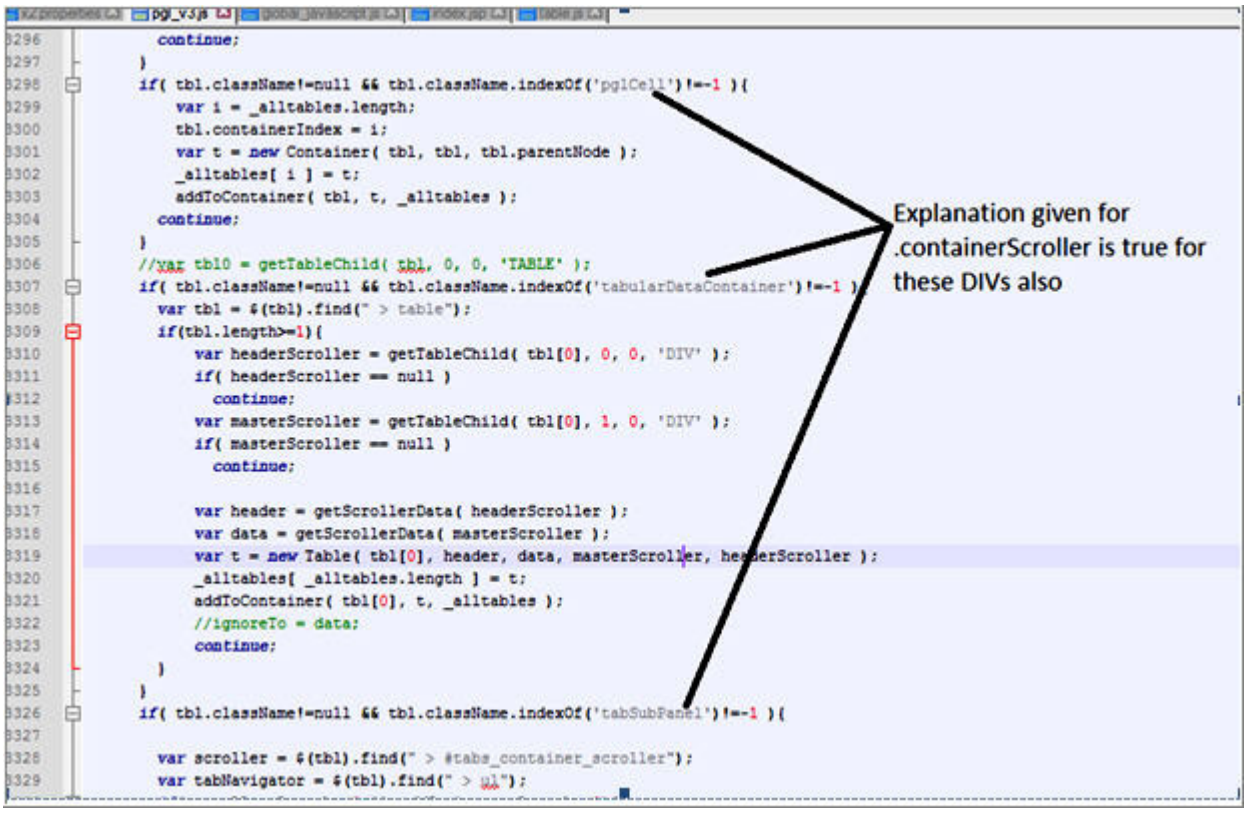
Scroll bar width/height is maintained with JavaScript scrollHelper() function. This function exist in old UI and it is modified to handle new HTML.

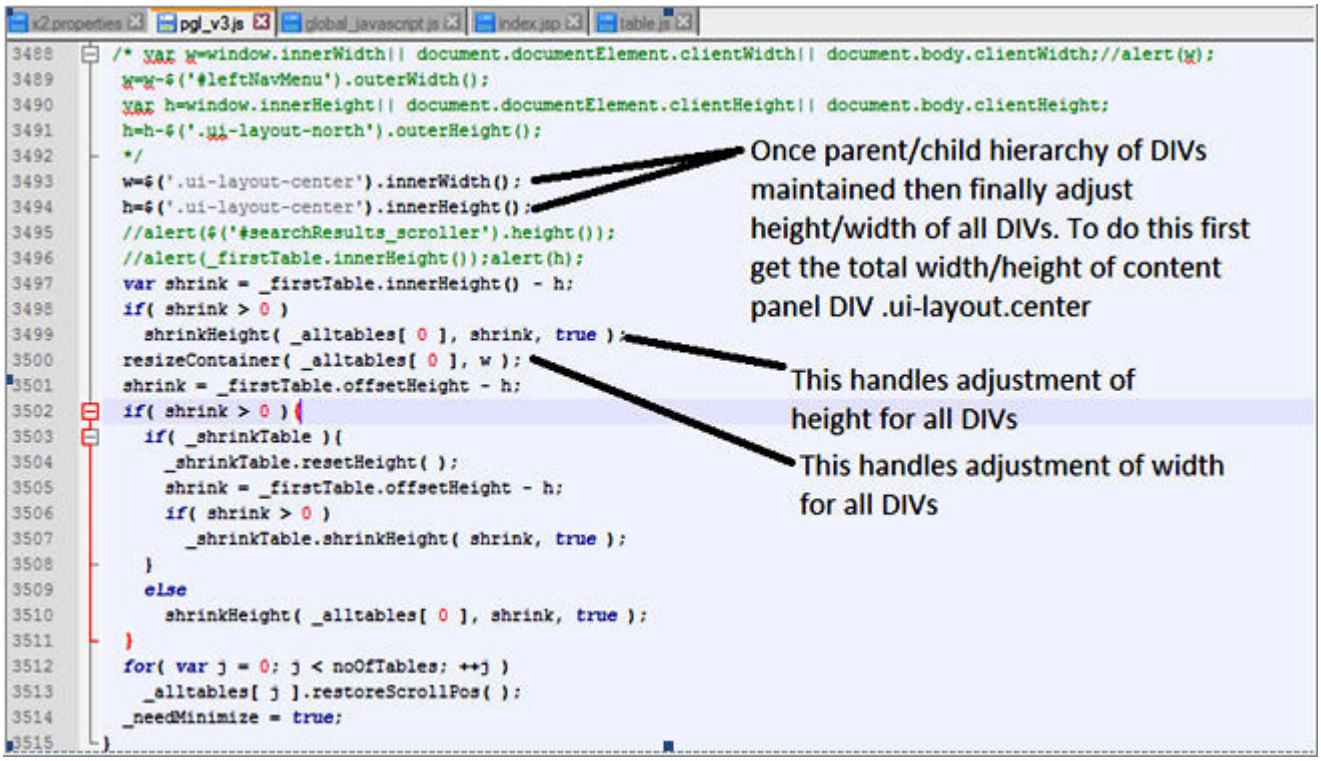
Below elements require scroll bars:

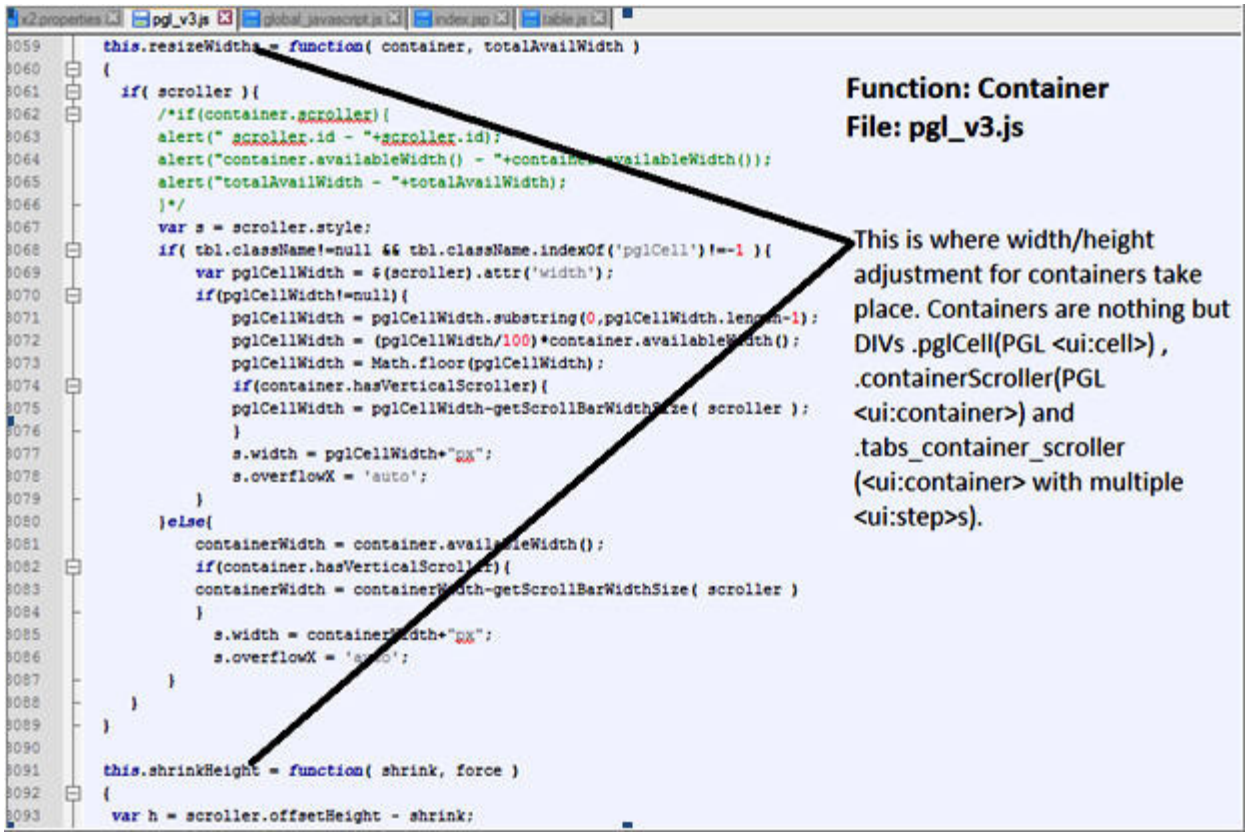
- DIV .containerScroller - equals to PGL <ui:container>
- DIV .pglCell - equals to PGL <ui:cell>
- DIV .tabularDataContainer - equals to PGL <ui:table>
- DIV .tabs_container_scroller - equals to PGL <ui:container> with more than one <ui:step> inside it.

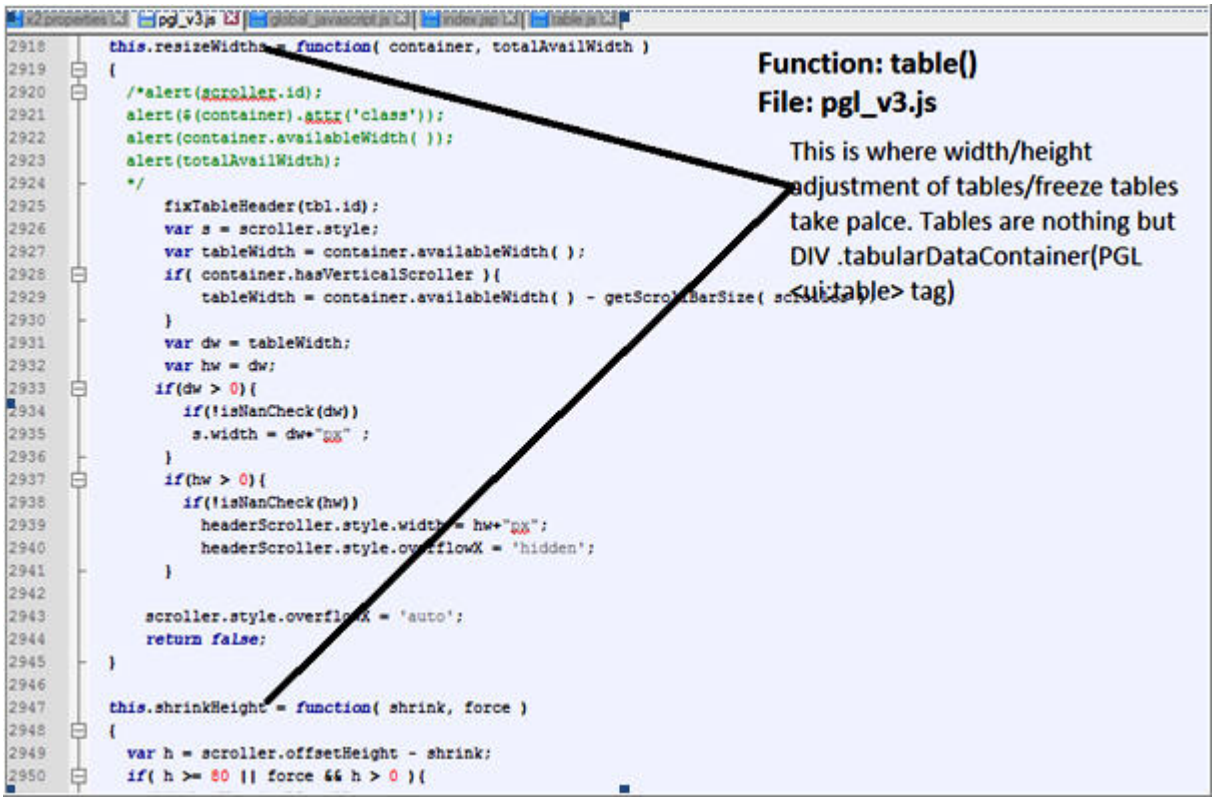
- Two DIVs .tabularDataContainer arranged in single <tr> - it is Freeze table and equals to PGL <ui:table> with attributes 'freeze'/'defaultFreeze' enabled.




Explanation given for .containerScroller is true for these DIVs also


Once parent/child hierarchy of DIVs maintained then finally adjust height/width of all DIVs. To do this first get the total width/height of content panel DIV .ui-layout.center
This handles adjustment of height for all DIVs
This handles adjustment of width for all DIVs


Function: Container
File: pgl_v3.js
This is where width/height adjustment for containers take place. Containers are nothing but DIVs .pglCell(PGL <ui:cell>) , .containerScroller(PGL <ui:container>) and .tabs_container_scroller (<ui:container> with multiple <ui:step>s).


Function: table()
File: pgl_v3.js
This is where width/height adjustment of tables/freeze tables take palce. Tables are nothing but DIV .tabularDataContainer(PGL <ui:table> tag)

# PGL Changes

PGL developers need to perform below changes if .pgl files are not working.

- **No document.location for forwarding to another page/URL**

  Use MDM API loadMainContentPanel() to forward to another URL

  **Example:**

  **Old Code**

  ```
  document.location = omxContextPath + &quot;/
  view.x2ps?START_WORKFLOW=ManageActivity&amp;SOLUTION_NAME=MDM&amp
  ;SERVICE_NAME=ADMIN_UI&quot;;
  ```

  **New Code**

  ```
  loadMainContentPanel(omxContextPath + &quot;/
  view.x2ps?START_WORKFLOW=ManageActivity&amp;SOLUTION_NAME=MDM&amp
  ;SERVICE_NAME=ADMIN_UI&quot;,'appFrame');
  ```

- **No document.write for including files dynamically**

  Use jQuery append() for including files

  **Example:**

  **Old Code**

  ```
  document.write('&lt;link rel=&quot;stylesheet&quot;
  type=&quot;text/css&quot;  href=&quot;bcm/wfeditor/css/
  nodes.css&quot; />');
  ```

  **New Code**

  - Add DIV in PGL <ui:div id="html_div"/>
  - Use jQuery .append() to add files to DIV

  ```
  $('#html_div').append('&lt;link rel="stylesheet" href="../../bcm/
  wfeditor/css/nodes.css" type="text/css" /&gt;');
  ```

- **Specify 'width' attribute for PGL <ui:cell>**

  It is good practice to specify 'width' attribute to <ui:cell>. Otherwise PGL framework will not assume any width for that <ui:cell> and width of <ui:cell> expands to the width of content inside it.

  **Examples:**

  1. If width specified i.e. <ui:cell width="20%">

  **Old UI**

  - Width adjustment is not in control of PGL framework and it is in browser control.
  - If content inside <ui:cell> require more than 20% width then browser expands <ui:cell> to the width of content.
  - If it may increase more than 20% and it may affect other elements inside whole UI. But developer intention is to have always width 20% irrespective of content width inside <ui:cell>. This is one of main reason why scroll bars were not working well in some of old UIs.

**New UI**

- First it gets the width of parent container. Parent may be <ui:container>/<ui:cell>. Let us say width of parent is 800px. Then it calculates width 20% of 800px that is 160px.
- Finally it sets 160px to <ui:cell>. Setting width in pixels always works and it is independent of content inside <ui:cell> and this is what developer wants.

2. If width not specified that is <ui:cell>

   In this case old/new UI behaves in same way. Width of <ui:cell> expands to the width of content inside it. This may not work well if content inside <ui:cell> is bigger.

3. If width 100% specified that is <ui:cell width="100%">

**Old UI**

- Width of <ui:cell> expands to the width of content inside it. This may not work well if content inside <ui:cell> is bigger.

**New UI**

- First it gets the width of parent container. Parent may be <ui:container>/<ui:cell>. Let us say width of parent is 800px. Then it calculates width 100% of 800px that is 800px.
- It then removes parent vertical scroll bar width (if any) and it is usually around 16px.Removal of vertical scroll bar width is necessary to make child vertical scroll bar visible.
- Finally it sets 784px to <ui:cell>. Setting width in pixels always works and it is independent of content inside <ui:cell> and this is what developer wants.

- **Specify 'omxContextPath' in document.form.action for static pages**

  **Old UI**

  ```
  document.form.action="/upload_kit/uploads/
  local_upload_file.jsp?TEMPLATE_NAME=" + "EntityCode_SetTemplate"
  +"&amp;FILE_PATH=" + fileName + "&amp;DATA_TYPE=" +ext +
  "&amp;DISPLAY_NAME=" + "Code_Set";
  ```

  **New UI**

  ```
  document.form.action=omxContextPath+"/upload_kit/uploads/
  local_upload_file.jsp?TEMPLATE_NAME=" + "EntityCode_SetTemplate"
  +"&amp;FILE_PATH=" + fileName + "&amp;DATA_TYPE=" +ext +
  "&amp;DISPLAY_NAME=" + "Code_Set";
  ```

  **Note:**

  No changes required if URL refers to servlet mapping.

  Example:

  Below URL refers to servlet as *.x2ps is mapped to servlet in web.xml.

  ```
  var url =
  'start.x2ps?START_WORKFLOW=HierarchyDesigner&amp;SERVICE_NAME=UI_
  WORKFLOW&amp;SOLUTION_NAME=MDM&amp;' + selId
  +'&amp;POPUP=true&amp;editHier=true';
  var winObj=
  window.open(url,'_blank',&quot;channelmode=no,fullscreen=no,direc
  ```

```
tories=no,location=no,menubar=no,resizable=yes,scrollbars=yes,sta
tus=no,titlebar=no,toolbar=no,width=800,height=700&amp;quot;);
```

- **Help Files: Specify relative path correctly for static pages**

  **Old UI**

  ```
  <ui:help displayPage="bcm/help/Edit_Hierarchy_help.htm" />
  ```

  **New UI**

  ```
  <ui:help displayPage="../../bcm/help/Edit_Hierarchy_help.htm" />
  ```

  **Note:**

  No changes required if URL is generated dynamically.

  Example:

  ```
  <ui:help displayPage="{$pages:ManageCodeSetHelp}"/>
  ```

- **Dropdown Issues**

  Dropdowns are converted to searchable dropdowns using select2 plugin http://ivaynberg.github.io/select2/

  Below are possible issues:

  - Issue1

    If dropdown options added/removed/updated dynamically through Ajax then it will not immediately reflect in searchable dropdown. There should be call to MDM JS function enableSelect2() to reflect changes in searchable dropdown but this issue is fixed at framework level. No action required for custom workflows.

  - Issue 2

    If there is no Ajax call to manipulate dropdown options but there is custom JavaScript which changes dropdown options. In this case PGL developers should call enableSelect2() manually at the end of such changes.

    **Note:** Override JavaScript enableSelect2() as empty function inside specific UI to disable searchable dropdown.

- **Specify scrollable="true" if width/height/scrollbar not correct**

  ```
  <ui:container scrollable="true">
  //Some PGL code
  </ui:container>

  <ui:table scrollable="true">
  //Some PGL code
  </ui: table>
  ```

  **Note:** If attribute 'scrollable' not specified or scrollable="false" then MDM displays that element as is that is width/height/scrollbars are not adjusted for that PGL element.

- **Do not override JavaScript showBody() inside .pgl files**

  - There will be few modifications in JavaScript showBody() in every release. It is good practice not to override this function inside .pgl files.

- Instead override function calls inside showBody(). For example you may override onLoad().

- **Custom JavaScript changes**

  Any custom JavaScript written targeting old HTML/CSS may not work as expected in new UI. New HTML/CSS follows HTML5/CSS3 standards.

  **PGL developers need to change their custom JavaScript to target new HTML5/CSS3.**

# CHAPTER 4 Basic Controls

## What's In This Chapter

This chapter gives you information on basic controls in PGL.

Topics include:


- Introduction
- Container
- Step
- Search and Results
- Text Area
- Button
- Button Group
- Check Box
- Radio Button
- Display Field
- Chart Field
- Entry Field
- Data List
- Hidden Field
- Field Group
- Field Space
- Divider
- Drop Down
- Option
- Horizontal Rule (Hr)
- Vertical Rule (Vr)
- Header
- Help
- Validation
- Date
- Compare
- Page Group
- Chart
- Root

- Custom Validator
- For Each
- Login
- Navigation
- Non-Breaking Space
- Pad
- Pad Item
- Sort
- Image
- Image Link Field
- Link
- Link Field
- Links
- URL
- Table
- Treecell
- IFrame

# Introduction

Every component described in this chapter provides the following details:

- Description: gives you a brief description of the component.
- Syntax: gives you the syntax of the component.
- Example: gives you an example of how the component can be used.
- Example Data: gives the example data for the component.
  - Data from Existing Example: gives the data from the existing example for the component.
  - Data from Example Project: gives the data from the ExampleProject available with the Teradata Studio.

  **Note:** Input variable usually $inXml is set in preprocessing of the UI node. Output variable refers to $outXml is set in postProcessing of the UI node.These variables can be checked by using the debugger or by printing the values. Alternatively, the variables can be user defined and accessed using debugger or by printing on the console.
- Required Attributes: list of required attributes of the component.
- Optional Attributes: list of optional attributes of the component.
- Content: gives the list of components that a component can contain.
- Used By: gives the list of other components that can use this component.

# Container

All PGL documents must have at least one frame which acts as a container for a group of PGL-UI Steps (Panels). A container can contain one or more panels. The container element has no intrinsic adornments like title bar and other additional attributes.

The following are the three types of containers:

1. Simple container: has only one PGL step (a simple panel).
2. Tabbed Container. has multiple PGL steps (multiple panels inside a frame).
3. Wizard: has multiple steps presented like a wizard.

## Syntax

The following is the syntax for container:

```
<ui:container>
```

## Examples

Example 1:

The following PGL fragment shows a visible container with context help:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry"
source="studio.controller:load">

      <ui:container id="Container1" name="Container1">
      <ui:help displayPage="{$pages:fields_help}"/>
      <ui:step displayText="PGL Step or Simply a frame" id="Step1"
      name="Step1"/>

  </ui:container>
</ui:page-group>
```

Figure 4: Simple Container

The container element supports the following:

- It can contain a context help with the use of a ui:help element.
- For a wizard or tabbed container at least two steps must be present.
- A multiple step document with a container attribute type declared as Wizard is presented as a Wizard and attribute type declared as Tab is presented as a Tabbed Container.
- It supports scrolling with its use of a Boolean attribute Scrollable.

Example 2:

The following is an example of a tabbed container.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container scrollable="true">
      <ui:step id="op" displayText="Order Point">
         <ui:field-group>
            <ui:display-field displayText="Name" name="NAME"
            value="Teradata Technologies"/>
            <ui:display-field displayText="Address 1" name="ADDRESS1"
            value="11511 Luna Road"/>
            <ui:display-field displayText="City" name="CITY"
            value="Dallas"/>
            <ui:display-field displayText="State" name="STATE"
            value="Texas"/>
            <ui:display-field displayText="Country" name="COUNTRY"
            value="USA"/>
         </ui:field-group>
      </ui:step>
<ui:step id="bp" displayText="Bill Point">
      <ui:field-group>
         <ui:display-field displayText="Name" name="NAME"
         value="Teradata Technologies"/>
         <ui:display-field displayText="Address 1" name="ADDRESS1"
         value="11511 Luna Road"/>
         <ui:display-field displayText="City" name="CITY"
         value="Dallas"/>
         <ui:display-field displayText="State" name="STATE"
         value="Texas"/>
         <ui:display-field displayText="Country" name="COUNTRY"
         value="USA"/>
      </ui:field-group>
</ui:step>

</ui:container>

</ui:page-group>
```

The following figure shows a tabbed container with two steps.

Figure 5: Tabbed Container

## Example Data

Data is not required for this component.

## Required Attributes

The following are the required attributes for a container.

Table 1: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |

## Optional Attributes

The following are the optional attributes for a container.

Table 2: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| collapsable | Set to true if container can be collapsable. Default value is false. |
| collapsed | |
| displayCondition | Conditional that determines if the button should be displayed or not. Default value is true. |
| displayText | Text to be displayed on the title bar of the container. Automatically internationalized. |
| editable | Set to true if container should allow editable components. Background will be shown in light blue if set to true. Default value is false. |
| height | Sets the height of the container to n pixels or a percentage of the available space. |
| inner | Set to true if container is to be rendered as an inner container. Usually the title bar background is shown in a lighter shade. Default value is false. |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| selectedStep | An XPath expression used for setting STEP_ ID for a wizard. |
| showFavorite | |
| type | Specifies the type of the container. Possible values include 'Wizard' or 'Tabs'. Default value is “Tabs” |
| width | Sets the width of the container to n pixels or a percentage of the available space. Default value is 100%. |

## Content

ui:header, ui:help, ui:step

## Used By

ui:cell, ui:page-group, ui:step

# Step

A step is a panel which contains a group of PGL—UI elements. It provides the ability to embed arbitrary collection of presentation components. The step is based on its parent—the container element.

The step is either a tab in a tabbed container or a step in a wizard container. You must have at least one step within every container. If only one step is defined for a tabbed container, the container defaults to a simple container.

The step has a title bar and the displayText attributes that define what is displayed in the title. The step element also supports scrolling with use of the scrollable attribute.

## Examples

The following examples show the three styles of step element with the minimum subset of attributes.

Example 1:

The following is an example of using a simple step inside a simple container.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container>

      <ui:step id="Step1" name="Step1" displayText="A Simple Frame and a
      Simple Panel"/>

</ui:container>
</ui:page-group>
```

The following figure displays the simple step within a simple container.

Figure 6: Simple Step

Example 2:

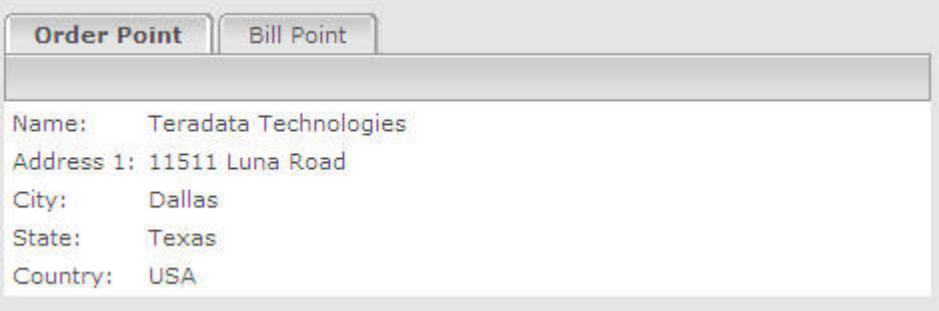
The following is an example of using a step inside a tabbed container in non-editable mode.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container scrollable="true">
      <ui:step id="op" displayText="Order Point">
         <ui:field-group>
            <ui:display-field displayText="Name" name="NAME"
```

```
                value="Teradata Technologies"/>
                <ui:display-field displayText="Address 1" name="ADDRESS1"
                value="11511 Luna Road"/>
                <ui:display-field displayText="City" name="CITY"
                value="Dallas"/>
                <ui:display-field displayText="State" name="STATE"
                value="Texas"/>
                <ui:display-field displayText="Country" name="COUNTRY"
                value="USA"/>
            </ui:field-group>
        </ui:step>
        <ui:step id="bp" displayText="Bill Point">
        </ui:step>
</ui:container>
</ui:page-group>
```

The following figure displays the step within a tabbed container in non-editable mode.

Figure 7: Tabbed Step



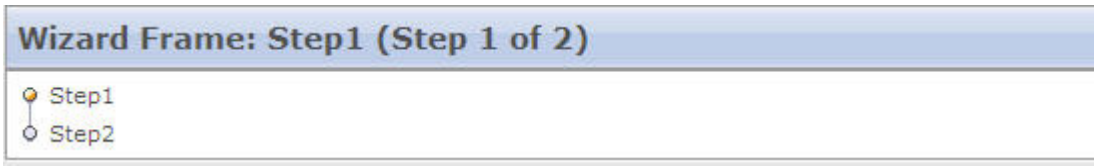

Example 3:

The following is an example of using a step in series, as steps of a wizard.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container displayText="Wizard Frame" id="Container1" type="Wizard">
        <ui:step id="Step1" name="Step1" displayText="Step1"/>
        <ui:step id="Step2" name="Step2" displayText="Step2"/>
</ui:container>

</ui:page-group>
```

The following figure shows the steps in series as steps of a wizard.

Figure 8: Steps




## Example Data

Workflow: BasicControls

Step: Wizard

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
   <SAMPLE_XML>
      <STEP_ID Value="step1"/>
   </SAMPLE_XML>
</TO_DOCVAR>
```

Set the attribute selectedStep on the container as "`{$root/STEP_ID/@Value}`" Set the `STEP_ID` value in your data according to the business logic to define the flow between the wizard steps.

## Required Attributes

The following are the required attributes for a step.

Table 3: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |

## Optional Attributes

The following are the optional attributes for a step.

Table 4: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the button should be displayed or not. Default value is true. |
| displayText | Text to be displayed on the title bar of the container. Automatically internationalized. |
| editable | Set to true if container should allow editable components. Background will be shown in light blue if set to true. Default value is false. |
| onClick | Specify the javascript to execute if the field is clicked. |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| selectedStep | An XPath expression used for setting STEP_ ID for a wizard. |

### Used By

ui:container

### Content

ui:buttons, ui:chart, ui:container, ui:dynamic-pgl, ui:field-group, ui:grid, ui:header, ui:help, ui:horizontal-tree, ui:hr, ui:pivot, ui:script, ui:search, ui:table, ui:validation, ui:vertical-tree, ui:workbook

# Search and Results

Search is a convenience container for handling standard search functionality. A combination of PGL elements can deliver powerful results with the help of the simple view hierarchy in a PGL document. One such combination is the Search and Table PGL elements combined to deliver a Saveable search frame with Sortable, Filterable and Pageable results.

### Syntax

The following is the syntax for Search:

```
<ui:search>
```

### Examples

The following are the examples with a combination of sortable, filterable, and pageable results.

Example 1:

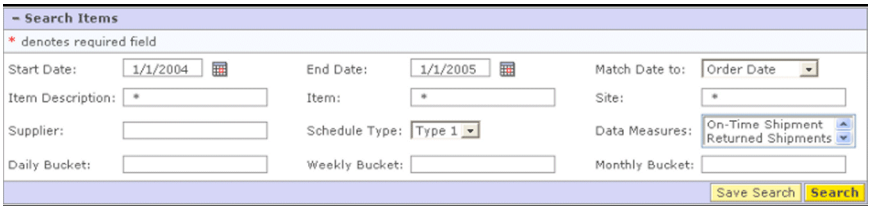
The following example illustrates search frame before the results.

```
<ui:search id="search1" displayText="Search Items">
  <ui:field-group columns="3" >
     <ui:entry-field size="10" value="1/1/2004" name="DATEFROM"
     dataType="Date" displayText="Start Date"/>
     <ui:entry-field size="10" value="1/1/2005" name="DATETO"
     dataType="Date" displayText="End Date"/>
     <ui:dropdown optionSrc="$root/DATETYPES/*" displayText="Match
     Date to" id="DateType" />
     <ui:entry-field displayText="Item Description"  value="*"
     id="Item_Desc" name="Item_Desc" />
     <ui:entry-field displayText="Item"  value="*" id="Item"
     name="Item" />
     <ui:entry-field displayText="Site"  value="*" id="Site"
     name="Site" />
     <ui:entry-field displayText="Supplier" id="Supplier"
     name="Supplier" />
```

```
        <ui:dropdown displayText="Schedule Type" optionSrc="$root/
        SCHEDULETYPES/*" id="SType" />
        <ui:dropdown displayText="Data Measures" size="2"
        optionSrc="$root/DATAMEASURES/*" id="DMSure"/>
        <ui:entry-field displayText="Daily Bucket" id="DBucket"
        name="DBucket" />
        <ui:entry-field displayText="Weekly Bucket" id="WBucket"
        name="WBucket" />
        <ui:entry-field displayText="Monthly Bucket" id="MBucket"
        name="MBucket" />
    </ui:field-group>
</ui:search>
```

The following figure illustrates the preceding code.

Figure 9: Search frame before the results




Example 2:

The following example illustrates search frame with results (combining search and table elements).

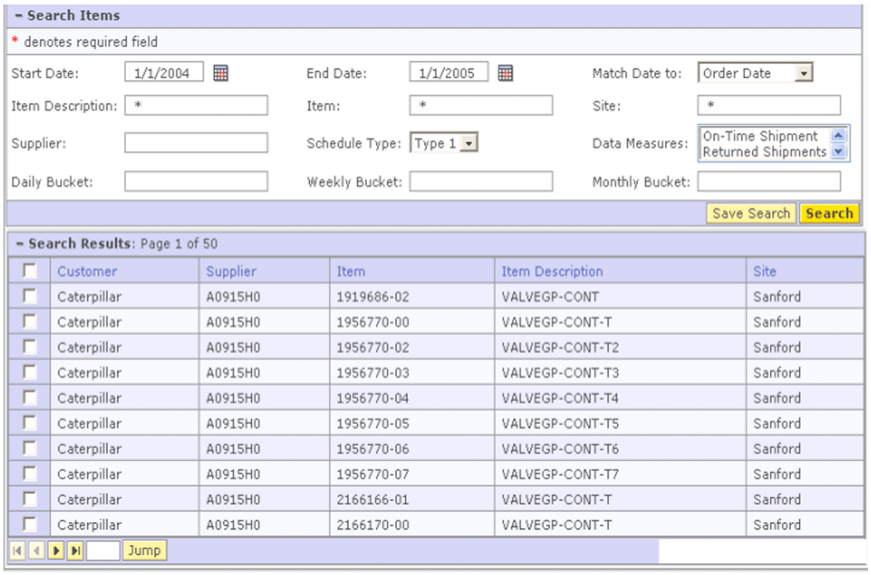
```
<ui:container>
<ui:step id="Outer" displayText=" ">
<ui:search id="search1" displayText="Search Items">
<ui:field-group columns="3" >
  <ui:entry-field size="10" value="1/1/2004"
  name="DATEFROM" dataType="Date" displayText="Start
  Date"/>
  <ui:entry-field size="10" value="1/1/2005"
  name="DATETO" dataType="Date" displayText="End
  Date"/>
  <ui:dropdown optionSrc="$root/DATETYPES/*"
  displayText="Match Date to" id="DateType" />
  <ui:entry-field displayText="Item Description"
  value="*" id="Item_Desc" name="Item_Desc" />
```

```
  <ui:entry-field displayText="Item"  value="*"
  id="Item" name="Item" />
  <ui:entry-field displayText="Site"  value="*"
  id="Site" name="Site" />
  <ui:entry-field displayText="Supplier" id="Supplier"
  name="Supplier" />
  <ui:dropdown displayText="Schedule Type"
  optionSrc="$root/SCHEDULETYPES/*" id="SType" />
  <ui:dropdown displayText="Data Measures" size="2"
  optionSrc="$root/DATAMEASURES/*" id="DMSure"/>
  <ui:entry-field displayText="Daily Bucket"
  id="DBucket" name="DBucket" />
  <ui:entry-field displayText="Weekly Bucket"
  id="WBucket" name="WBucket" />
  <ui:entry-field displayText="Monthly Bucket"
  id="MBucket" name="MBucket" />
</ui:field-group>
</ui:search>
<ui:table  displayText="Search Results" displayTextNoRecords="Not
found" rowId="ROW_ID" select="$root/CAT/ITEMS" scrollable="false"
selectable="All" collapsable="true" >
<ui:field-group >
  <ui:display-field sortable="true"
  displayText="Customer" size="20" name="Customer"/>
  <ui:display-field sortable="true"
  displayText="Supplier" size="20" name="Supplier"/>
  <ui:display-field sortable="true" displayText="Item"
  name="Item"/>
  <ui:display-field sortable="true" displayText="Item
  Description" name="Item_desc"/>
  <ui:display-field sortable="true" displayText="Site"
  name="Site"/>
</ui:field-group>
```

```
</ui:table>

</ui:step>

</ui:container>
```

The following figure illustrates the preceding code.

Figure 10: Search frame with results



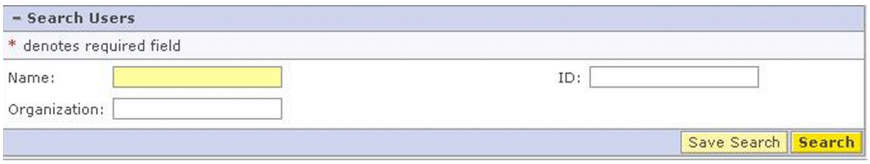

Example 3:

The following example of using a search container to populate a table.

```
<ui:search displayText="Search Users">

<ui:field-group columns="2">

<ui:entry-field displayText="Name" name="NAME" editable="true"/>

<ui:entry-field displayText="ID" name="ID" editable="true"/>

<ui:entry-field displayText="Organization" name="ORG_NAME"
editable="true"/>

</ui:field-group>

</ui:search>
```

The following figure illustrates the preceding code.

Figure 11: Search Container to populate a table



Example 4:

The following example illustrates as how to achieve Save Search on PGL Favorites:

```
<TO_DOCVAR AssignToVar="finalFilter">
<A/>
</TO_DOCVAR>
<IF_TEST Test="$thisParam/BUTTON_ID/@Value='search_button' or
$thisParam/BUTTON_ID/@Value='createFavourite'">
<THEN>
<FORM_SEARCH_FILTER Value="{$thisParam}" AssignToVar="filters">
<attributeSetID MatchBy="CONTAINS"/>
<updatedLogicalTableName MatchBy="CONTAINS"/>
</FORM_SEARCH_FILTER>
              <FOR_EACH SelectList="$filters/FILTER"
CurrentElement="thisFilter">
<APPEND_TO_XML SelectList="$finalFilter">
<TO_XML SelectList="$thisFilter/*"/>
</APPEND_TO_XML>
</FOR_EACH>
</THEN>
</IF_TEST>
<GET_DOCUMENT Name="AttributeSet" ServiceName="BCM_MASTER"
AssignToVar="res" ReturnRowCount="yes" DirtyRead="true">
<TO_XML SelectList="$finalFilter/*"/>
</GET_DOCUMENT>
```

## Example Data

Refer to search example in ExampleProject in Teradata Studio.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Search.

Table 5: Optional Attributes

| Attribute | Description |
|---|---|
| action | Specify the URL that will process the form. This is usually an x2 controller command. |

Table 5: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| anyFieldHasErrors | Not Used |
| anyFieldIsRequired | Not Used |
| clearSearch | |
| collapsable | Set to true if container can be collapsable. Default value is false. |
| collapsed | |
| displayCondition | Conditional that determines if the button should be displayed or not. |
| displayText | Text to be displayed as the title of the container. Automatically internationalized. |
| exportSearch | |
| help | URL link to location of help page. |
| id | Identifier for HTML component. |
| method | Specify the parameter-passing style. Possible values include 'GET' and 'POST'. Default value is 'POST'. |
| name | Identifier for HTML component. Used in order to address the container within javascript code. |
| saveSearch | |
| target | Define the name of the frame or window to receive the referenced document. |

### Used By

ui:cell, ui:page-group, ui:step

### Content

ui:buttons, ui:field-group, ui:validation

# Text Area

A Text Area is an HTML component that defines a multiline text input area.

### Syntax

The following is the syntax for Text Area:

```
<ui:textarea>
```

## Example

The following is an example of using a Text Area:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
<ui:container>
      <ui:step id="Step1" name="Step1" displayText="Fields">
      <ui:field-group columns="2">
         <ui:display-field displayText="Text Field" name="NAME"
         value="Data"/>
         <ui:entry-field displayText="Editable Text Field" name="NAME"
         value="Data" editable="true"/>
         <ui:checkbox displayText="Check Box" name="BOX" value="true"/>
         <ui:radiobutton displayText="Radio button 1" id="radio1"
         name="RADIO" value="true"/>
         <ui:radiobutton displayText="Radio button 2" id="radio2"
         name="RADIO"/>
         <ui:textarea id="textArea" name="TEXT_AREA"
displayText="Textarea" rows="4"/>

      </ui:field-group>
      </ui:step>
</ui:container>
</ui:page-group>
```

The following figure shows a Text Area field within a simple container.

Figure 12: Text Area

## Example Data

### Data from Existing Example

The following is an example of the input variable for the Text Area:

```
<ROOT>

  <TEXT_AREA Value=""/>

</ROOT>
```

The following is an example of the output variable for the Text Area that captures user entered value:

```
<PARAMETERS>

   <TEXT_AREA Value="User entered this value"/>
```

```
<PARAMETERS>
```

### Data from Example Project

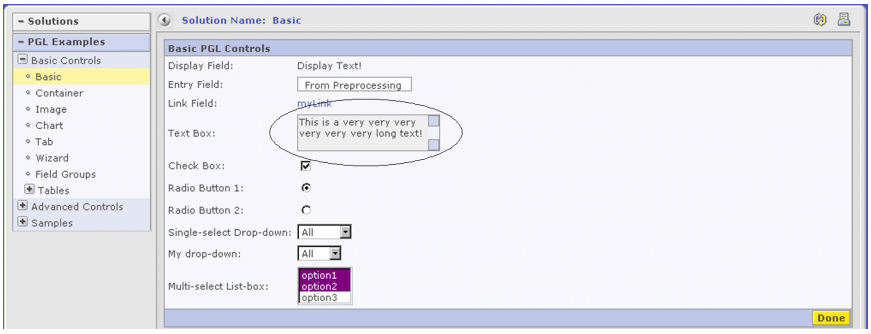
Workflow: BasicControls

Step: Basic

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
<SAMPLE_XML>
<TEXT_BOX Value="This is a very very very very very very long text!"/
>
</SAMPLE_XML>
</TO_DOCVAR>
```

Figure 13: Text Area Example




## Required Attributes

The following are the required attributes for Text Area.

Table 6: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. |

## Optional Attributes

The following are the optional attributes for Text Area.

Table 7: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| columns | Minimum number of columns the fields should be divided between. |
| configurable | Set to true if field can be configurable. The table must be configurable for the field to be configurable. Default is true. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| disabled | Set to true if field should be disabled. Default value is false. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| editable | Set to true if field should be editable. Default value is true. |
| filter | Set to true if field can be filtered. The table must be filtered for the field to be filtered. Default value is true. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type of the header. |
| headerEditable | Set to true if header should be editable. Default value is false. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |

Table 7: Optional Attributes

| Attribute | Description |
|---|---|
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| headerUom | Sets the unit of measure at the header level. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| massEntry | Set to true if field can be mass entry. Default value is true. |
| numericPattern | If set to "0000000000", you can insert only the numbers and get rid of the decimal separator. |
| onBlur | |
| onClick | Specify the javascript to execute if the field is clicked. |
| onFocus | |
| onKeyDown | |
| onKeyUp | |
| preferredWidth | |
| primaryKey | |
| readOnly | |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| rows | Minimum number of rows the fields should be divided between. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortType | |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| title | Text to displayed as the title of textarea. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |

Table 7: Optional Attributes

| Attribute | Description |
|---|---|
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| uploadType | Sets the upload type for the field. Set to “fil” if the upload item is file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| visibility | |
| displayPage | URL of the page that will be transitioned to without posting the form. |
| elementName | Possible value is T_FIELD_ENTRY. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| inlineEdit | Sets the input or text area fields to appear like display fields. Hover on these fields, the fields appear like normal input or text area. |

### Content

ui:compare-validator, ui:custom-validator, ui:links, ui:url

### Used By

ui:field-group, ui:otherwise, ui:when

# Button

Button creates an action component that fires an event.

### Syntax

The following is the syntax for button:

```
<ui:button>
```

### Example

The following is an example of using a button:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container>
      <ui:step id="c1" displayText="Buttons" editable="true">
         <ui:field-group columns="1">
            <ui:entry-field displayText=" Text Field" name="NAME"
            value="Data" editable="true"/>
         </ui:field-group>
      <ui:buttons>
         <ui:button displayText="Secondary 1"/>
         <ui:divider/>
      <ui:button displayText="Secondary 2"/>
 <ui:button displayText="Primary" name="PRIMARY" emphasized="true"/>
      </ui:buttons>
      </ui:step>
</ui:container>
</ui:page-group>
```

The following figure shows three buttons. The primary button is emphasized.

Figure 14: Button

The following is the example Post action for the given button

```
<IF_TEST Test="$userAction='Primary'">

<THEN>

<!—Write your logic here-->

</THEN>

</IF_TEST>
```

Figure 15: Define condition expression using userAction

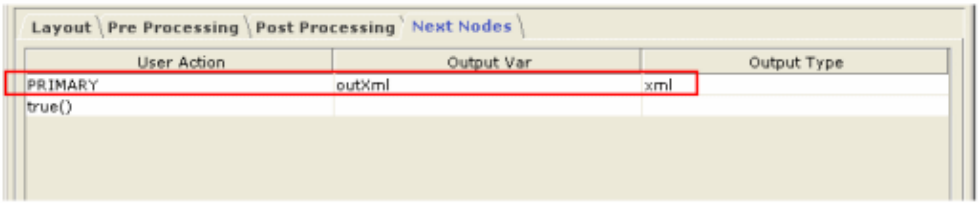
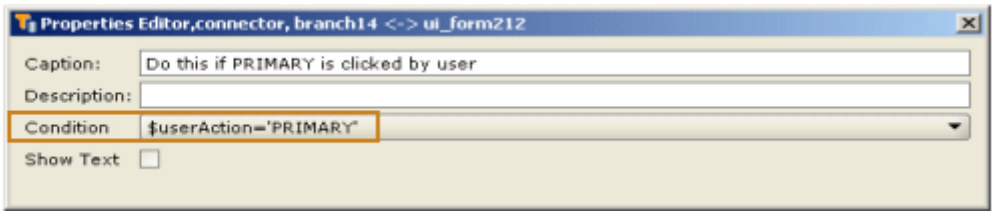


Figure 16: Attach condition to connector




Button click triggers next nodes from the UI form. The userAction variable stores the name of the button clicked by the user.

Next node condition expressions can be defined using the userAction variable and attached to the connector for the next node or self connector to the same node

### *Data from Example Project*

Workflow: BasicControls

Step: Basic

PreProcessing:

```
<IF_TEST Test="$userAction='NEXT'">
   <THEN>
      <TO_DOCVAR AssignToVar="outXml">
         <SAMPLE_XML>
            <STEP_ID Value="step2"/>
         </SAMPLE_XML>
      </TO_DOCVAR>
   </THEN>
</IF_TEST>
```

## Required Attributes

The following are the required attributes for button.

Table 8: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Name of the button. Used in order to address the button within javascript code, input and output variables. |

## Optional Attributes

The following are the optional attributes for button.

Table 9: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| disabled | Displays the button in the disabled state if set to true. Prevents the user from using the button. |
| displayCondition | Conditional that determines if the button should be displayed or not. |
| displayPage | URL of the page that will be transitioned to without posting the form. |
| displayText | Text to be displayed on the button. Automatically internationalized. |
| emphasized | Displays the button as 'highlighted' if set to true. |
| enabledBySelections | Button is set to disabled until a table row is selected. Default value is true. |
| onClick | Specify the javascript to execute if the button is clicked. |
| submitAsXML | Number of fraction digits to be displayed. |
| submitTo | Name of the x2 command that the form will be posted to. |
| target | The target frame or window that the action will be posted to. |
| targetName | Define the name of the frame or window to receive the referenced document. |
| targetType | Specify the type of frame or window to direct the referenced document. The possible value is 'popup'. |
| validate | Determines if validation should be performed or not. |

### Content

ui:script, validate

### Used By

ui:buttons, ui:field-group, ui:otherwise, ui:row, ui:links, ui:when

# Button Group

Button group is a container for holding a group of buttons.

### Syntax

The following is the syntax for button group:

```
<ui:buttons>
```

## Example

The following is an example of using a button group:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">

<ui:container>
      <ui:step id="c1" displayText="Buttons" editable="true">
         <ui:field-group columns="1">
            <ui:entry-field displayText=" Text Field" name="NAME"
            value="Data" editable="true"/>
         </ui:field-group>
      <ui:buttons>
         <ui:button displayText="Secondary 1"/>
         <ui:divider/>
      <ui:button displayText="Secondary 2"/>
 <ui:button displayText="Primary" name="PRIMARY" emphasized="true"/>
      </ui:buttons>
      </ui:step>
</ui:container>
</ui:page-group>
```

Figure 17: Button Group

## Example Data

Data is not required for this component.

## Required Attributes

The following are the required attributes for button group.

Table 10: Required Attributes

| Attribute | Description |
|---|---|
| Id | Identifier for HTML component. Used in order to address the button within javascript code. |

## Optional Attributes

The following are the optional attributes for button group.

Table 11: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Condition that determines if the components should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Condition that determines if the component should be displayed or not. |
| display text | Text to be displayed on the button. Automatically internationalized. |
| select | An Xpath expression used for adding dynamic options. The value is automatically evaluated as an expression. |

### Content

ui:button, ui:divider

### Used By

ui:cell, ui:page-group, ui:search, ui:step, ui:table, ui:chart

# Check Box

A check box is an HTML component that can be in either an ‘on’ (true) or ‘off ’ (false) state. Clicking on a check box toggles its state.

### Syntax

The following is the syntax for check box:

```
<ui:checkbox>
```

### Example

The following is an example of using a check box:

```
<ui:field-group columns="2">
  <ui:display-field displayText="Text Field" name="NAME"
  value="Data"/>
  <ui:entry-field displayText="Editable Text Field" name="NAME"
  value="Data" editable="true"/>
  <ui:checkbox displayText="Check Box" name="BOX" value="true"/>
  ...
</ui:field-group>
```

The following figure shows the checkbox field within a simple container.

Figure 18: Check Box

## Example Data

### *Data from Existing Example*

The following is an example of the input variable for the check box:

```
<ROOT>
   <BOX Value="true"/>
</ROOT>
```

The following is an example of the output variable for the Check Box that captures its final state:

```
<PARAMETERS>
   <BOX Value="true"/>
<PARAMETERS>
```

### *Data from Example Project*

Workflow: BasicControls

Step: Basic

PreProcessing:

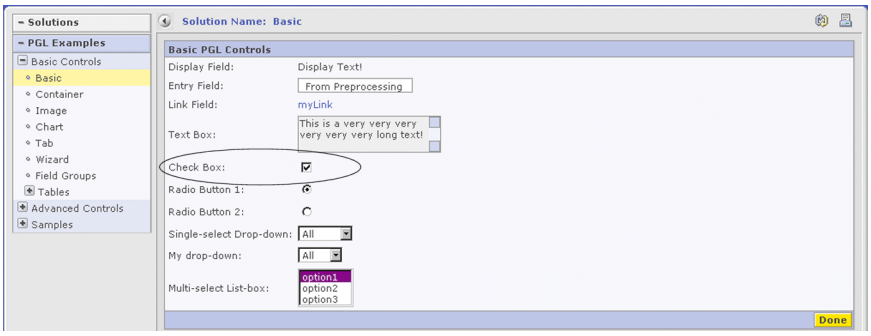
```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
           <CHECK_BOX Value="true"/>
     </SAMPLE_XML>
</TO_DOCVAR>
```

PostProcessing:

To access the value of the checkbox in post processing use the following

```
<SET Var="MyCheckbox" FromValue="{$outXml/CHECK_BOX/@Value}"/>
```

Figure 19: Check Box Example




## Required Attributes

The following are the required attributes for check box.

Table 12: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. Used in order to address the button within javascript code, input and output variables. |

## Optional Attributes

The following are the optional attributes for check box.

Table 13: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| checkedValue | Specifies the value submitted if box is checked. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| configurable | Set to true if field can be configurable. The table must be configurable for the field to be configurable. Default is true. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |

Table 13: Optional Attributes

| Attribute | Description |
|---|---|
| defaultValue | Default value used if no value is initially given. |
| disabled | Set to true if field should be disabled. Default value is false. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Sets the URL for a link. Note: attribute is deprecated. |
| displayPage | URL of the page that will be transitioned to without posting the form. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| editable | Set to true if field should be editable. Default value is true. |
| elementName | Possible value is T_FIELD_ENTRY. |
| filter | Set to true if field can be filtered. The table must be filtered for the field to be filtered. Default value is true. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type for the header. Used for internationalization. |
| headerEditable | Set to true if the header should be editable. Default value is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| headerUom | Sets the unit of measure at the header level. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| massEntry | Set to true if field is mass entry. Default value is true. |
| numericPattern | If set to "0000000000", you can insert only the numbers and get rid of the decimal separator. |
| onClick | Specify the javascript to execute if the field is clicked. |
| preferredWidth | |
| primaryKey | |

Table 13: Optional Attributes

| Attribute | Description |
|---|---|
| readOnly | |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortType | |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| title | Text to displayed as the title of checkbox. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| uploadType | Sets the upload type for the field. Set to “fil” if the upload item is file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| visibility | |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

ui:compare-validator, ui:custom-validator, ui:links, ui:url

# Radio Button

A radio button is an HTML component that can be in either an ‘on’ (true) or ‘off’ (false) state. Clicking on a radio button selects the item and unselects other options in its group.

## Syntax

The following is the syntax for radio button:

```
<ui:radiobutton>
```

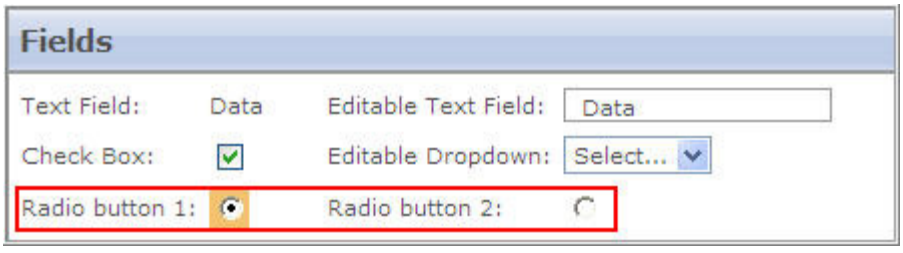
## Example

The following is an example of using Radio Button.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
<ui:container>
 <ui:step id="c1" displayText="Fields">
 <ui:field-group columns="2">
      <ui:display-field displayText="Text Field" name="NAME"
      value="Data"/>
      <ui:entry-field displayText="Editable Text Field" name="NAME"
      value="Data" editable="true"/>
      <ui:checkbox displayText="Check Box" name="BOX" value="true"/>
      <ui:dropdown displayText="Editable Dropdown" selectOne="true">
         <ui:option id="1" value="One"/>
         <ui:option id="2" value="Two"/>
         <ui:option id="3" value="Three"/>
      </ui:dropdown>
      <ui:radiobutton displayText="Radio button 1" id="radio1"
      name="RADIO" value="true" checkedValue="radio1"/>
      <ui:radiobutton displayText="Radio button 2" id="radio2"
      name="RADIO" checkedValue="radio2"/>
      </ui:field-group>
 </ui:step>
</ui:container>
</ui:page-group>
```

The following figure displays the group of radio buttons within a simple container.

Figure 20: Radio Button




The button to be selected by default is specified in the radio button definition. In the above example, radio button 1 is selected by default by setting its value to “true”.

## Example Data

### *Data from Existing Example*

The following is an example of the output variable for the radio button that captures its final state:

```
<PARAMETERS>
   <RADIO Value="true"/>
<PARAMETERS>
```

In this example, user selected radio button with id=radio1.

If the checkedValue attribute is used, that value is assigned to the radio button component in the output variable.

For example,

```
<ui:radiobutton displayText="radiotest1" id="radio1" name="radio1"
checkedValue="radio1"/>
<PARAMETERS>
   <radio1 Value="radio1"/>
</PARAMETERS>
```

### *Data from Example Project*

Workflow: BasicControls

Step: Basic

PreProcessing:

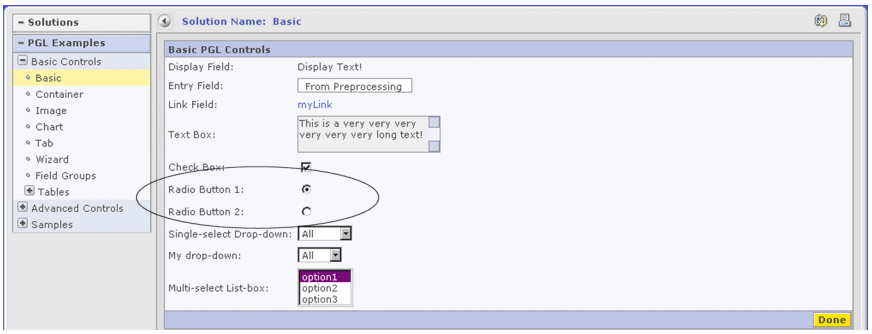
```
<TO_DOCVAR AssignToVar="inXml">
   <SAMPLE_XML>
      <RADIO_BTN Value="selectedFirst"/>
   </SAMPLE_XML>
</TO_DOCVAR>
```

PostProcessing:

To access the value of the radio button in post processing use the following

```
<SET Var="MyRadioButton" FromValue="{$outXml/RADIO_BTN/@Value}"/>
```

Figure 21: Radio Button Example




## Required Attributes

Table 14: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. Used in order to address the button within javascript code, input and output variables. |

## Optional Attributes

Table 15: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| checkedValue | Specifies the value submitted if radio button is selected. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| configurable | Set to true if field can be configurable. The table must be configurable for the field to be configurable. Default is true. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| disabled | Set to true if field should be disabled. Default value is false. |

Table 15: Optional Attributes

| Attribute | Description |
|---|---|
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | Sets the URL for a link. Note: Attribute is deprecated. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| editable | Set to true if field should be editable. Default value is true. |
| elementName | Possible value is T_FIELD_ENTRY. |
| filter | Set to true if field can be filtered. The table must be filtered for the field to be filtered. Default value is True. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type for the header. Used for internationalization. |
| headerEditable | Set to true if header should be editable. Default value is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| headerUom | Sets the unit of measure at the header level. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| massEntry | Set to true if field can be mass entered. The table must be mass entry for the field to be mass entry. Default value is true. |
| numericPattern | If set to "0000000000", you can insert only the numbers and get rid of the decimal separator. |
| onClick | Specify the javascript to execute if the field is clicked. |
| preferredWidth | |
| primaryKey | |
| readOnly | |

Table 15: Optional Attributes

| Attribute | Description |
|---|---|
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortType | |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| title | Text to displayed as the title of radio button. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| uploadType | Sets the upload type for the field. Set to “fil” if the uploaded item is a file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| visibility | |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

ui:compare-validator, ui:links, ui:url

# Display Field

It is the field component that displays its label and value in read-only mode.

## Syntax

The following is the syntax for display field.

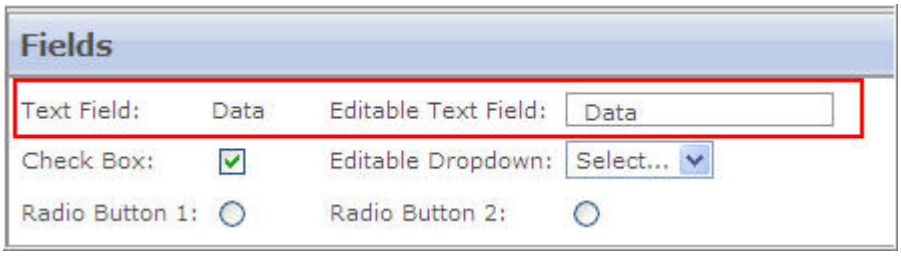
```
<ui:display-field>
```

## Example

Example 1: The following is an example of using a display-field within a simple container.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
<ui:container>
 <ui:step id="c1" displayText="Fields">
 <ui:field-group columns="2">
      <ui:display-field displayText="Text Field" name="NAME"
      value="Data"/>
      <ui:entry-field displayText="Editable Text Field" name="NAME"
      value="Data" editable="true"/>
      <ui:checkbox displayText="Check Box" name="BOX" value="true"/>
      <ui:dropdown displayText="Editable Dropdown" selectOne="true">
         <ui:option id="1" value="One"/>
         <ui:option id="2" value="Two"/>
         <ui:option id="3" value="Three"/>
      </ui:dropdown>
      <ui:radiobutton displayText="Radio button 1" id="radio1"
      name="RADIO" value="true" checkedValue="radio1"/>
      <ui:radiobutton displayText="Radio button 2" id="radio2"
      name="RADIO" checkedValue="radio2"/>
      </ui:field-group>
 </ui:step>
</ui:container>
</ui:page-group>
```

Figure 22: Display Field




Example 2: To display a decimal column value on PGL

For decimal display on PGL, that is, for Decimal_<m>_<n>, where m = Total number of digits, n= number of digits after precision and m-n = number of digits before precision, that is to display a decimal column value on PGL for Decimal_12_10, total number of digits = 12, number of digits after precision = 10 and number of digits before precision = 2, use the following syntax.

```
<ui:display-field id="DecimalColumn" name="DecimalColumn"
displayText="DECIMAL_COLUMN" />
```

Default behaviour is as defined by its datatype:

- If the value is 2.232, then it will display 2.2320000000 by automatically appending zeros for precision field.
- If the value is 0 (i.e -0.0000000000/0.0000000000), then it will display exponential form as (0E-10)

To customize the decimal display on the PGL UI (that is, to display or restrict precision up to n digits after decimal point), specify datatype=”Number” and decimals="n" attributes on <ui:display-field> component. Where n = digits after precision

```
<ui:display-field id="DecimalColumn" name="DecimalColumn"
displayText="DECIMAL_COLUMN" dataType="Number" decimals="3" />
```

- If the value is 2.2320000000, then it will display 2.232

## Example Data

### *Data from Existing Example*

The following is an example of the input variable for the display field:

```
<ROOT>
  <NAME Value="Data"/>
</ROOT>
```

The following is an example of the output variable for the display field:

```
<PARAMETERS>
  <NAME Value="Data"/>
<PARAMETERS>
```

### *Data from Example Project*

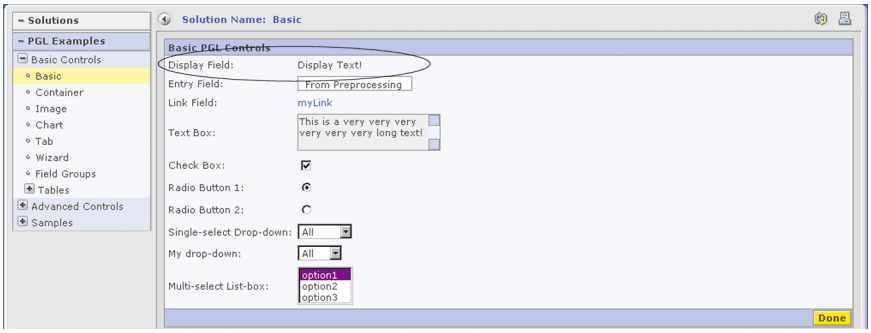
Workflow: BasicControls

Step: Basic

Data can be specified in the PGL directly by using the value attribute as follows

```
<ui:display-field displayText="Display Field" name="DISPLAY_FIELD"
value="Display Text!"/>
```

Figure 23: Display Field Example




## Required Attributes

Table 16: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. Used in order to address the field within javascript code, input and output variables. |

## Optional Attributes

Table 17: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| configurable | Set to true if field can be configurable. The table must be configurable for the field to be configurable. Default is true. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |

Table 17: Optional Attributes

| Attribute | Description |
|---|---|
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | Sets the URL for a link. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| filter | Set to true if field can be filtered. The table must be filtered for the field to be filtered. Default value is true. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type of the header. |
| headerEditable | Set to true if header should be editable. Default value is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| headerUom | Sets the unit of measure at the header level. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| massEntry | Set to true if field can be mass entry. Default value is true. |
| maxlength | Maximum number of characters allowed in entry-field. |
| onClick | Specify the javascript to execute if the field is clicked. |
| preferredWidth | |
| primaryKey | |
| readOnly | |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |

Table 17: Optional Attributes

| Attribute | Description |
|---|---|
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortType | |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| src | URL link to the location of image. |
| style | Specify a style controlling the appearance of cell's contents. |
| targetType | Specify the type of frame or window to direct the referenced document. The possible value is 'popup'. |
| title | Text to displayed as the title of display field. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| uploadType | Sets the upload type for the field. Set to “fil” if the upload item is file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| visibility | |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

ui:compare-validator, ui:links, ui:url

# Chart Field

It is the field component used within a chart tag for indicating the chart measures.

## Syntax

The following is the syntax for chart field.

```
<ui:chart-field>
```

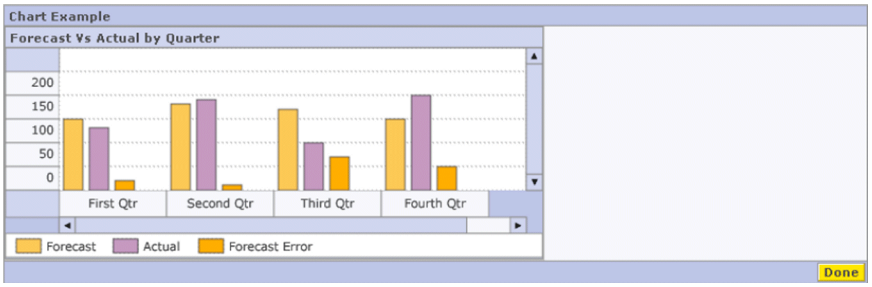
## Example

The following is an example of a bar chart field within a chart component.

```
<ui:chart width="750" height="200" xLabelsWidth="175"
yAxisNumTicks="5" displayText="Planning" select="$root/DATA/
FORECASTS">
   <ui:chart-field displayText="" name="FROM_DATE" dataType="Date"/>
   <ui:chart-field displayText="Forecast" name="SCHEDULE_QTY"/>
   <ui:chart-field displayText="BOH" name="BEGINING_ON_HAND"/>
   ...
</ui:chart>
```

The following figure shows six chart fields within a chart.

Figure 24: Chart Field




## Example Data

See the example data in the section "Chart" on page 105.

## Required Attributes

None.

## Optional Attributes

Table 18: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| callbackData | Sets the data to be return in the callback. Note: Attribute is deprecated |
| color | Color of the chart field. It should be specified as hexadecimal like #ff0000 for red. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , 'DateTime' and 'Currency'. Default value is 'Text'. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed beside chart field's legend. Automatically internationalized. |
| name | Name of the chart field. |
| select | An XPath expression used for evaluating the value of the chart field. |
| type | Type determines the way the chart field is displayed. Types can either be line or bar. Default is bar. |
| value | Value of the field. |

## Used By

ui:chart

## Content

ui:buttons,ui:chart-field

# Entry Field

It is the field component that receives text input.

## Syntax

The following is the syntax for entry field.

```
<ui:entry-field>
```

## Example

The following is an example of using an entry field for text data within a simple container.

Example 1:

```
<ui:field-group columns="2">
  <ui:display-field displayText="Text Field" name="NAME"
  value="Data"/>
  <ui:entry-field displayText="Editable Text Field" name="NAME"
  value="Data" editable="true"/>
  ...
</ui:field-group>
```

The following figure illustrates the preceding code.

Figure 25: Entry Field

Example 2:

The following is an example of using an entry field for File data type. For each of column of type “file”, you should declare entry field of type “file” in the PGL template as below:

```
<ui:entry-field id="EmpData" name="EmpData" displayText="Employee
Data(Attachment)" dataType="file"/>
```

For each field of type File, two implicit variables will be created in the output variable of the PGL Node (Ex: pglFormOutput) namely $fieldName_FileName and $fieldName_Contents which will have the filename and file contents respectively.

To save the file data information into the database, a new rule “insertFileDataType” is defined in the BPE_META service. You have to pass the file information along with column and table name, using which this rule will add an entry into SYS_FILE_DATA and return the newly added record's ID value. This ID value should be passed to the ADD_DOC command as the value for the corresponding column of file type.

Example: Employee Document has property called EmpData of “file” type

```
<DO_TRANSACTION>
          <REQUEST Name="insertFileDataType" ServiceName="BPE_META"
AssignToVar="FileContentID">
            <FileName Value="{$pglFormOutput/EmpData_FileName/
@Value}"/>
            <FileContents Value="{$pglFormOutput/
EmpData_FileContents/@Value}"/>
            <TargetTableName Value="MST_EMPLOYEE"/>
            <TargetColumnName Value="EmpData"/>
          </REQUEST>
```

```
          <IF_TEST Test="($FileContentID/RESULT/@STATUS =
'SUCCESS')">
            <THEN>
             <ADD_DOC_SMART Name="Employee" ServiceName="BCM_MASTER">
                <EmpId Value="{$pglFormOutput/EmpId/@Value}"/>
                <EmpName Value="{$pglFormOutput/EmpName/@Value}"/>
                <DeptName Value="{$pglFormOutput/DeptName/@Value}"/>
                <EmpData Value="{$FileContentID/RESULT/ID/@Value}"/>
              </ADD_DOC_SMART>
        </THEN>
            </IF_TEST>
       </DO_TRANSACTION>
```

## Example Data

### *Data from Example Project*

Workflow: BasicControls

Step: Basic

PreProcessing:

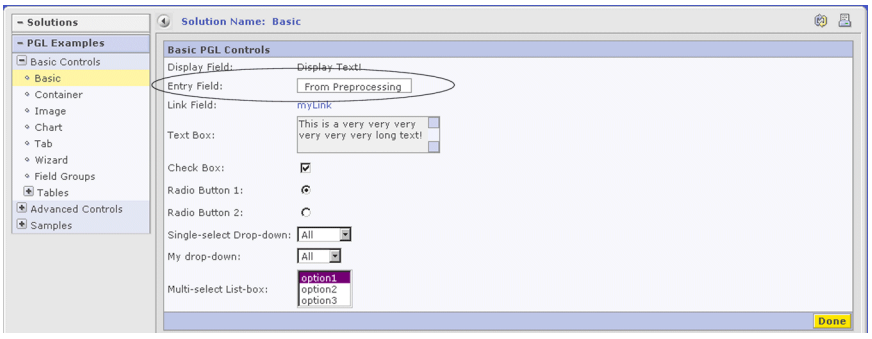
```
<TO_DOCVAR AssignToVar="inXml">
<SAMPLE_XML>
<ENTRY_FIELD Value="From Preprocessing"/>
</SAMPLE_XML>
</TO_DOCVAR>
```

PostProcessing:

To access the value of the radio button in post processing use the following

```
<SET Var="MyEntryField" FromValue="{$outXml/ENTRY_FIELD/@Value}"/>
```

Figure 26: Entry Field Example



## Required Attributes

Table 19: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. Used in order to address the field within javascript code, input and output variables. |

## Optional Attributes

Table 20: Optional Attributes

| Attribute | Description |
|---|---|
| maxlength | Maximum number of characters allowed in entry-field. |
| onChange | Specify the javascript to execute if the entry field text is changed. |
| disabled | Set to true if field should be disabled. Default value is false. |
| editable | Set to true if field should be editable. Default value is true. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| configurable | Set to true if field can be configurable. The table must be configurable for the field to be configurable. Default value is true. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’, ‘File’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | Sets the URL for the link. Note: |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |

Table 20: Optional Attributes

| Attribute | Description |
|---|---|
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type for the header. |
| headerEditable | Set to true if the header should be editable. Default is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| massEntry | Set to true if field can be mass entry. The table must be mass entry for the field to be mass entry. Default value is true. |
| numericPattern | If set to "0000000000", you can insert only the numbers and get rid of the decimal separator. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |

Table 20: Optional Attributes

| Attribute | Description |
|---|---|
| uploadType | Sets the upload type for the field. Set to “fil” if the uploaded item is a file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| list | The list attribute refers to the <ui:datalist> element that contains pre-defined options for an <ui:entry-field> element. |
| inlineEdit | Sets the input or text area fields to appear like display fields. Hover on these fields, the fields appear like normal input or text area. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |

## Used By

ui:field-group, ui:otherwise, ui:when, ui:pivot

## Content

ui:compare-validator, ui:custom-validator, ui:links, ui:url

## Placeholder Attribute

The Placeholder is an new attribute to PGL <ui:entry-field>. The placeholder attribute specifies a short hint that describes the expected value of an input field (for example: a sample value or a short description of the expected format).

The short hint is displayed in the input field before you enter a value.

**Note:** The Place Holder attribute is an HTML 5 feature and works only in PGL3 with IE10+ and Firefox latest versions.

### Syntax

The following is the syntax for <Placeholder> tag.

```
<ui:entry-field id="Placeholder" name="Placeholder" displayText="First
Name" placeholder="First Name"/>
```

### Example

The following are the examples of using <placeholder> tag.

Example 1:

```
<ui:field-group>
            <ui:entry-field id="Placeholder" name="Placeholder"
displayText="First Name" placeholder="First Name"/>
</ui:field-group>
```

The following figure illustrates the preceding code.

Figure 27: Placeholder Attribute Example

# Data List

The <ui:datalist> tag is always combined with <ui:entry-field>.The <ui:datalist> tag specifies a list of pre-defined options for an < ui:entry-field> element. The <ui:datalist> tag provides “autocomplete” feature on <ui:entry-field> elements. You can see a drop-down list of pre-defined options as you input data. Use the < ui:entry-field> element's list attribute to bind it together with a <datalist> element.

The Data list feature is implemented differently in different browsers. For example, if you type letter “I”, IE10 shows options “starting with letter I” and Firefox shows options “containing letter I”

**Note:** The <ui:datalist> tag is an HTML 5 feature and works only in PGL3 with IE10+ and Firefox latest versions.

## Syntax

The following is the syntax for <ui:datalist> tag.

```
<ui:datalist unFormat="true" id="BrowsersList" name="BrowsersList">
```

## Example

The following are the examples of using <ui:datalist> tag.

Example 1: The below example uses <ui:option> to populate the datalist options.

```
<ui:field-group>
            <ui:entry-field id="Browsers" name="Browsers"
list="BrowsersList" displayText="Browsers"/>
            <ui:datalist unFormat="true" id="BrowsersList"
name="BrowsersList">
              <ui:option value="Internet Explorer"/>
              <ui:option value="Firefox"/>
              <ui:option value="Chrome"/>
              <ui:option value="Opera"/>
              <ui:option value="Safari"/>
            </ui:datalist>
```

```
          </ui:field-group>
```

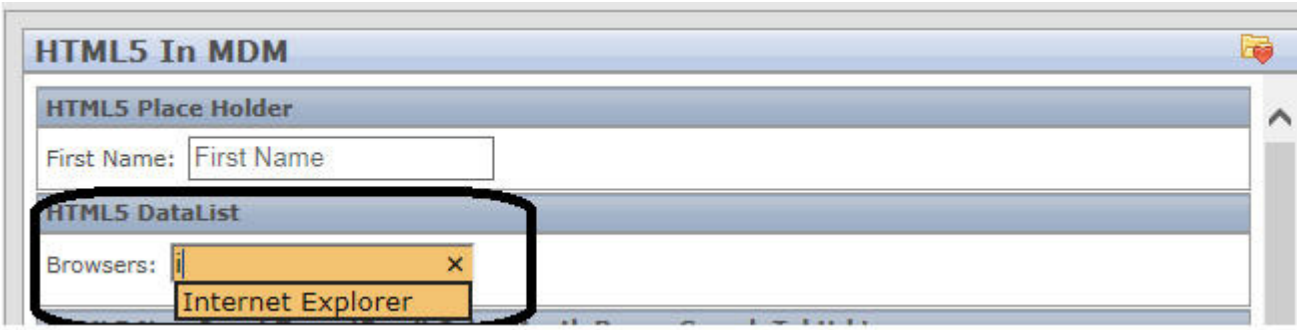
Value of <ui:datalist> "id" attribute and <ui:entry-field> "list" attribute should be same.

Example 2: Below example uses optionSrc attribute to populate the datalist options.

```
<ui:field-group>
            <ui:entry-field id="Browsers" name="Browsers"
list="BrowsersList" displayText="Browsers"/>
            <ui:datalist unFormat="true" id="BrowsersList"
optionSrc="$root/OPTIONS/OPTION" name="BrowsersList">
          </ui:datalist>
           </ui:field-group>
```

The below figure displays the datalist example.

Figure 28: Datalist Example




## Example Data

Data is not required for this component.

## Required Attributes

The following table lists the required attributes for UI:Datalist tag.

Table 21: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| Name | Specifies the parameter name that is submitted as part of the form. |

## Optional Attributes

The following table lists the optional attributes for UI:Datalist tag.

Table 22: Optional Attributes

| Attribute | Description |
|---|---|
| optionSrc | An X-path expression used to populate dropdown with options. |
| unFormat | Set to true for no i18nization. Default value is false. |

## Used By

ui:field-group

## Content

ui:option

# Hidden Field

It creates a hidden field for the page form.

## Syntax

The following is the syntax for hidden field.

```
<ui:hidden-field>
```

## Example

```
<ui:hidden-field name="Col2"/>
<ui:when test="Col2/@Value = 'Active'">
<ui:display-field name="Status" value="Active"/>
```

## Example Data

### *Data from Existing Example*

```
<ROOT>
  <Col2 Value="Active"/>
</ROOT>
```

### *Data from Example Project*

Workflow: Samples

Step: MassEntry

PGL:

```
<ui:hidden-field name="PROPERTY_NO" value="{NO/@Value}"/>
```

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
<SAMPLE_XML>
  <Response>
    <LINE>
      <NO Value="1"/>
```

```
        <VALUE Value="Value-1"/>
      </LINE>
    </Response>
</SAMPLE_XML>
```

PostProcessing:

To access the value of the hidden field in post processing use the following

```
<SET Var="MyHiddenField" FromValue="{$outXml/PROPERTY_NO/@Value}"/>
```

## Required Attributes

None.

## Optional Attributes

Table 23: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| name | Specify the name of the parameter that is submitted as part of the form. |
| onlyInHeader | When used in a table and set to true, the hidden variable will only be added once for the table and not for each row. Default value is false. |
| onlyJavascript | If set to true, the hidden field is only created as a javascript variable. |
| select | An XPath expression used for culling the data for the component's children. The value is automatically evaluated as an expression. |
| submitAsXML | Returns the data as an XML data structure. Default value is false. Note: Attribute is deprecated. |
| value | Initial value set to the field. |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

None.

# Field Group

It is the logical grouping of fields.

## Syntax

The following is the syntax for field group.

```
<ui:field-group>
```

## Examples

The following are the examples of using a field group.

Example 1:

The following is an example of a two column field group within a simple container.

```
<ui:container>
<ui:help displayPage="{$pages:fields_help}"/>
<ui:step id="c1" displayText="Fields" editable="true">
<ui:field-group columns="2">
<ui:display-field displayText="Text Field" name="NAME" value="Data"/
>
<ui:entry-field displayText="Editable Text Field" name="NAME"
value="Data" editable="true"/>
<ui:checkbox displayText="Check Box" name="BOX" value="true"/>
<ui:field-space displayText="Space"/>
<ui:dropdown displayText="Dropdown" editable="false" value="2">
  <ui:option id="1" value="One"/>
  <ui:option id="2" value="Two"/>
  <ui:option id="3" value="Three"/>
</ui:dropdown>
<ui:dropdown displayText="Editable Dropdown" selectOne="true">
  <ui:option id="1" value="One"/>
  <ui:option id="2" value="Two"/>
  <ui:option id="3" value="Three"/>
</ui:dropdown>
<ui:dropdown displayText="Editable Dropdown w/Src" value="2"
selectAll="true" optionSrc="OPTIONS/OPTION"/>
```

```
<ui:field-space displayText="Space"/>

<ui:radiobutton displayText="Radio button 1" id="radio1"
name="RADIO" value="true"/>

<ui:radiobutton displayText="Radio button 2" id="radio2"
name="RADIO"/>

</ui:field-group>

</ui:step>

</ui:container>
```

The following figure illustrates the preceding code.

Figure 29: Field Group

Example 2:

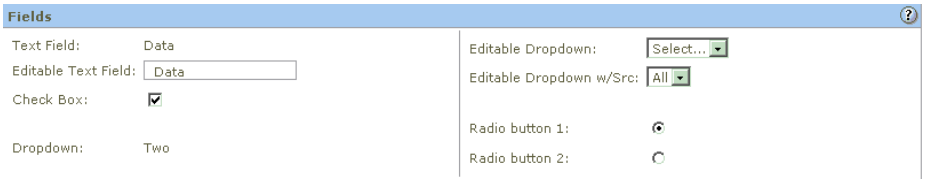
The following is an example of a nested field group within a simple container.

```
 <ui:container>

   <ui:help displayPage="{$pages:fields_help}"/>

   <ui:step id="c1" displayText="Fields" editable="true">

     <ui:field-group columns="2" showDivider="true">

       <ui:field-group>

         <ui:display-field displayText="Text Field" name="NAME"
value="Data"/>

         <ui:entry-field displayText="Editable Text Field"
name="NAME" value="Data" editable="true"/>

         <ui:checkbox displayText="Check Box" name="BOX"
value="true"/>

         <ui:field-space displayText="Space"/>

         <ui:dropdown displayText="Dropdown" editable="false"
value="2">

           <ui:option id="1" value="One"/>

           <ui:option id="2" value="Two"/>

           <ui:option id="3" value="Three"/>
```

```
          </ui:dropdown>
        </ui:field-group>
        <ui:field-group>
          <ui:dropdown displayText="Editable Dropdown"
selectOne="true">
            <ui:option id="1" value="One"/>
            <ui:option id="2" value="Two"/>
            <ui:option id="3" value="Three"/>
          </ui:dropdown>
          <ui:dropdown displayText="Editable Dropdown w/Src"
value="2" selectAll="true" optionSrc="OPTIONS/OPTION"/>
          <ui:field-space displayText="Space"/>
          <ui:radiobutton displayText="Radio button 1" id="radio1"
name="RADIO" value="true"/>
          <ui:radiobutton displayText="Radio button 2" id="radio2"
name="RADIO"/>
        </ui:field-group>
      </ui:field-group>
    </ui:step>
  </ui:container>
```

The following figure illustrates the preceding code.

Figure 30: Field Group




Example 3:

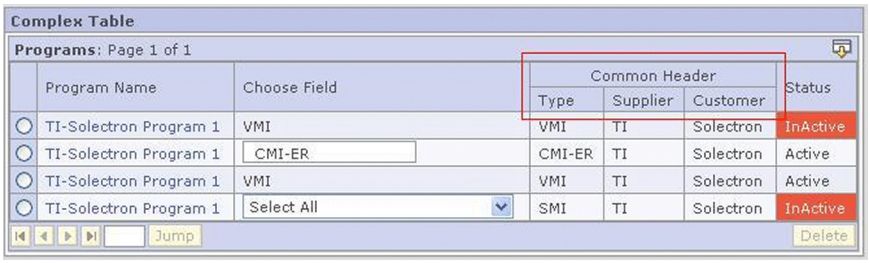
The following is an example of the multi-leveled field groups within a complex table.

```
<ui:table displayText="Programs" scrollable="false"
displayTextNoRecords="No Programs found" select="TPAS" rowSrc="TPA"
collapsable="false" selectable="Single">
<ui:help url="{$pages:search_results_help}"/>
<ui:field-group>
<ui:link-field name="TPA_NAME" displayText="Program Name">
```

```
<ui:url page="{$pages:cpm_program_review}">
  <TPA_ID select="TPA_ID/@Value"/>
</ui:url>
</ui:link-field>
...
<ui:field-group displayText="Common Header">
<ui:display-field displayText="Type" name="TPA_TYPE"/>
<ui:display-field displayText="Supplier" name="SUPP_NAME"/>
<ui:display-field displayText="Customer" name="CUST_NAME"/>
</ui:field-group>
<ui:display-field displayText="Status" name="STATUS" style="{STATUS/
@Style}"/>
</ui:field-group>
...
</ui:table>
```

The following figure illustrates the preceding code.

Figure 31: Field Group




## Example Data

Data is not required for this component.

## Required Attributes

None.

### Optional Attributes

Table 24: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| columns | Minimum number of columns the fields should be divided between. |
| displayCondition | Conditional that determines if the field group should be displayed or not. |
| displayText | Text to be displayed above the field group. Automatically internationalized. |
| filter | Sets the default for the field group to be filtered or not. Default value is true. |
| hideFields | Suppresses any data name contained in hideFields string value. |
| rows | Minimum number of rows the fields should be divided between. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| showDivider | If set true, shows horizontal dividers between child field-groups. |

### Used By

ui:cell, ui:field-group, ui:otherwise, ui:search, ui:step, ui:table, ui:when

### Content

ui:button, ui:checkbox, ui:choose-field, ui:date-range, ui:display-field, ui:dropdown, ui:entry-field, ui:field-group, ui:field-space, ui:hidden-field, ui:image-link-field, ui:image, ui:legend-field, ui:recent-entry-field, ui:link-field, ui:radiobutton, ui:textarea, ui:treecell

## Field Space

It is field component that maintains a space. Similar to a <td> </td> in HTML.

### Syntax

The following is the syntax for Image Link Field.

```
<ui:field-space>
```

### Example

The following is an example of using two field spaces to aid in positioning other fields within a field group.

```
<ui:field-group columns="2">
  <ui:display-field displayText="Text Field" name="NAME"
  value="Data"/>
  <ui:entry-field displayText="Editable Text Field" name="NAME"
  value="Data" editable="true"/>
  <ui:checkbox displayText="Check Box" name="BOX" value="true"/>
  <ui:field-space displayText="Space"/>
  <ui:dropdown displayText="Dropdown" editable="false" value="2">
     <ui:option id="1" value="One"/>
     <ui:option id="2" value="Two"/>
     <ui:option id="3" value="Three"/>
  </ui:dropdown>
  <ui:dropdown displayText="Editable Dropdown" selectOne="true">
     <ui:option id="1" value="One"/>
     <ui:option id="2" value="Two"/>
     <ui:option id="3" value="Three"/>
     </ui:dropdown>
  <ui:dropdown displayText="Editable Dropdown w/Src" value="2"
  selectAll="true" optionSrc="OPTIONS/OPTION"/>
  <ui:field-space displayText="Space"/>
  <ui:radiobutton displayText="Radio button 1" id="radio1"
  name="RADIO" value="true"/>
  <ui:radiobutton displayText="Radio button 2" id="radio2"
  name="RADIO"/>
</ui:field-group>
```

The following figure illustrates the preceding code.

Figure 32: Field Space

## Example Data

Data is not required for this component.

## Required Attributes

None

## Optional Attributes

Table 25: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

None.

# Divider

It renders a vertical separator between buttons in a button group.

## Syntax

The following is the syntax for divider.

```
<ui:divider>
```

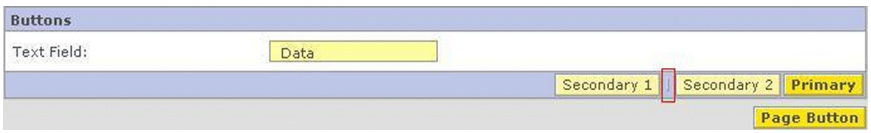
## Example

The following is an example of a divider within a button group.

```
<ui:buttons>
   <ui:button displayText="Secondary 1"/>
   <ui:divider/>
   <ui:button displayText="Secondary 2"/>
   <ui:button displayText="Primary" emphasized="true"/>
</ui:buttons>
```

The following figure illustrates the preceding code.

Figure 33: Divider



## Example Data

Data is not required for this component.

## Required Attributes

The following are the required attributes for divider.

Table 26: Required Attributes

| Attribute | Description |
|---|---|
| Id | Identifier for HTML component. Used in order to address the button within javascript code. |

## Optional Attributes

The following are the optional attributes for divider.

Table 27: Optional Attributes

| Attribute | Description |
|---|---|
| displayCondition | Conditional tat determines if the component should be displayed or not. |
| activity | Conditional that determines if the components should be displayed or not based on the user having permission for at least one of the listed activities. |
| display text | Text to be displayed on the button. Automatically internationalized. |

## Used By

ui:buttons

## Content

None.

# Drop Down

It is the field component that displays a multiple-choice menu or a scrolling list of options.

## Syntax

The following is the syntax for drop down.

```
<ui:dropdown>
```

## Example

The following is an example of using a dropdown with Select One activated, and another using XML data to populate its options.

```
<ui:field-group columns="2">
   <ui:display-field displayText="Text Field" name="NAME"
   value="Data"/>
   <ui:entry-field displayText="Editable Text Field" name="NAME"
   value="Data" editable="true"/>
   ...
   <ui:dropdown displayText="Editable Dropdown" selectOne="true">
      <ui:option id="1" value="One"/>
      <ui:option id="2" value="Two"/>
      <ui:option id="3" value="Three"/>
   </ui:dropdown>
   <ui:dropdown displayText="Editable Dropdown w/Src"
   name="DROP_DOWN" optionSrc="$root/OPTIONS/OPTION"/>
   ...
</ui:field-group>
```

The following shows two dropdown fields within a simple container.

Figure 34: Drop Down

## Example Data

### *Data from Existing Example*

The following is an example of the input variable for the dropdown that uses XML data to populate options:

```
<ROOT>
<OPTIONS>
<OPTION Value="One"/>
<OPTION Value="Two"/>
<OPTION Value="Three"/>
</OPTIONS>
</ROOT>
```

**Note:** The options have to be defined with a tag called `OPTION`. The parent tag (`OPTIONS` in the above example) can be user defined.

The following is an example of the input variable where the options are generated from a table:

```
<TO_DOCVAR AssignToVar="inXml">
  <ROOT>
     <OPTIONS>
  </ROOT>
</TO_DOCVAR>
<GET_DOCUMENT AssignToVar="options" Name="optionList"/>
<SET Var="optionsVar" FromSelect="$inXml/OPTIONS"/>
<FOR_EACH CurrentElement="currOption" SelectList="$options/option">
<TO_DOCVAR AssignToVar="currOptionVar">
  <ROOT>
     <OPTION Value="{$currOption/option/@Value}"/>
  </ROOT>
</TO_DOCVAR>
<APPEND_CHILDREN DocVar="optionsVar" FromDocVar="currOptionVar"/>
</FOR_EACH>
```

The above code generates the following inXml variable:

```
<ROOT>
  <OPTIONS>
     <OPTION Value="One"/>
     <OPTION Value="Two"/>
     <OPTION Value="Three"/>
  </OPTIONS>
</ROOT>
```

The following is an example of specifying the default value for the dropdown:

```
<ROOT>
  <DROPDOWN Value="One">
</ROOT>
```

The following is an example of the output variable for the dropdown when only one value is selected:

```
<PARAMETERS>
  <OPTION Value="One"/>
```

```
<PARAMETERS>
```

The following is an example of the output variable for the dropdown when multiple values are selected:

```
<PARAMETERS>
  <OPTION Value="One"/>
  <OPTION Value="Two"/>
<PARAMETERS>
```

The following is an example of the output variable for the dropdown when “All” is selected (this requires selectAll="true"):

```
<PARAMETERS>
  <OPTION Value=""/>
<PARAMETERS>
```

### *Data from Example Project*

Workflow: BasicControls

Step: Basic

PGL:

```
<ui:dropdown name="DROP_DOWN" selectAll="true" size="1"
displayText="Single-select Drop-down">
  <ui:option id="option1" value="option1"/>
  <ui:option id="option2" value="option2"/>
  <ui:option id="option3" value="option3"/>
</ui:dropdown>
<ui:dropdown name="LIST_BOX" size="3" displayText="Multi-select
List-box">
  <ui:option id="option1" value="option1"/>
  <ui:option id="option2" value="option2"/>
  <ui:option id="option3" value="option3"/>
</ui:dropdown>
```

PreProcessing:

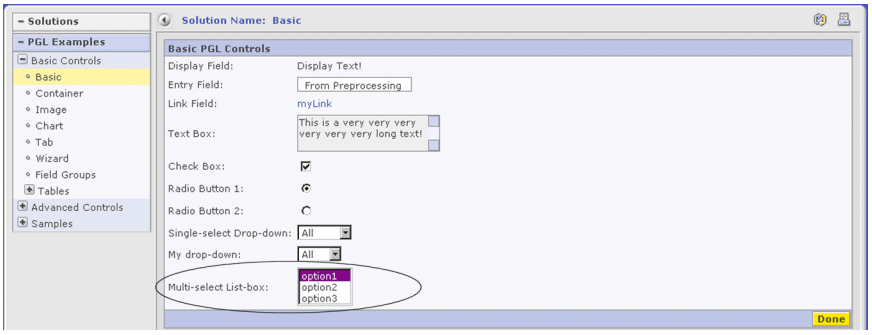
```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <LIST_BOX Value="option1"/>
      <LIST_BOX Value="option2"/>
    </SAMPLE_XML>
</TO_DOCVAR>
```

PostProcessing:

$outXml will have the following

```
<PARAMETERS>
<LIST_BOX Value="option1"/>
<LIST_BOX Value="option2"/>
</PARAMETERS>
```

Figure 35: Option Example




## Required Attributes

Table 28: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. Used in order to address the field within javascript code, input and output variables. |

## Optional Attributes

Table 29: Optional Attributes

| Attribute | Description |
|---|---|
| onChange | Specify that javascript to execute if a dropdown option is selected. See section on Script for syntax and examples. |
| optionSrc | An XPath expression used to populate dropdown with options. |
| selectAll | If set true, an option called “All” is added to the dropdown. This allows the user to select all options at once. |
| selectOne | If set true, the dropdown allows the user to only select one option at a time. |
| editable | Set to true if field should be editable. Default value is true. |

Table 29: Optional Attributes

| Attribute | Description |
|---|---|
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | Sets the URL for the link. Note: Attribute is deprecated. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Set to true if the header should be editable. Default is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |

Table 29: Optional Attributes

| Attribute | Description |
|---|---|
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the number of rows displayed. This should be set greater than one if user has to select multiple options. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:field-group, ui:otherwise, ui:when

### Content

ui:option, ui:compare-validator, ui:links, ui:url

# Option

It defines an option for a dropdown field.

### Syntax

The following is the syntax for Option.

```
<ui:option>
```

## Examples

The following is an example of using a dropdown field with static options.

```
<ui:field-group columns="2">
<ui:display-field displayText="Text Field" name="NAME" value="Data"/>
<ui:entry-field displayText="Editable Text Field" name="NAME"
value="Data" editable="true"/>
...
<ui:dropdown displayText="Editable Dropdown" selectOne="true">
<ui:option id="1" value="One"/>
<ui:option id="2" value="Two"/>
<ui:option id="3" value="Three"/>
</ui:dropdown>
...
</ui:field-group>
```

The following figure shows a dropdown field using static options within a simple container.

Figure 36: Option

## Example Data

See example in the "Drop Down" section.

## Required Attributes

The following are the required attributes for option.

Table 30: Required Attributes

| Attribute | Description |
|---|---|
| displayCondition | Conditional that determines if the component should be displayed or not. |
| value | Value to be displayed. If not specified, the id value is used instead. |
| selected | An Xpath expression used for adding dynamic options. The value is automatically evaluated as an expression. |
| activity | Conditional that determines if the components should be displayed or not based on the user having permission for at least one of the listed activities. |
| display text | Text to be displayed on the button. Automatically internationalized. |
| Id | Identifier for HTML component. Used in order to address the button within javascript code. |

### Optional Attributes

The following are the optional attributes for option.

Table 31: Optional Attributes

| Attribute | Description |
|---|---|
| id | Sets the identifier for this specific option. |
| value | Value to be displayed. If not specified, the id value is used instead. |

### Used By

ui:dropdown

### Content

None.

## Horizontal Rule (Hr)

It is a component that adds a horizontal ruler to the container.

### Syntax

The following is the syntax for horizontal rule.

```
<ui:hr>
```

### Example

The following is an example of a horizontal ruler used to separate the two field groups inside a container.

```
<ui:container editable="false">
<ui:header>
<ui:link displayText="Notes">
<ui:image src="/notes.gif"/>
</ui:link>
</ui:header>
<ui:step id="c3" displayText="Container 3">
<ui:field-group>
<ui:display-field displayText="Name" name="NAME" value="Sample Name"/>
</ui:field-group>
<ui:hr/>
<ui:field-group columns="2">
<ui:display-field displayText="Label 1" name="DATA_1" value="More Data"/
>
<ui:display-field displayText="Label 2" name="DATA_2" value="Even More
Data"/>
</ui:field-group>
</ui:step>
```

```
</ui:container>
```

The following figure illustrates the preceding code.

Figure 37: Horizontal Rule

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

None.

## Used By

ui:step

## Content

None.

# Vertical Rule (Vr)

It is a component that adds a vertical ruler to the container.

## Syntax

The following is the syntax for vertical rule.

```
<ui:vr>
```

## Required Attributes

None.

## Optional Attributes

None.

## Used By

ui:cell, ui:row

## Content

None.

## Example Data

Data is not required for this component.

# Header

It is a container for holding image links for the title area of a ui:container.

## Syntax

The following is the syntax for header.

```
<ui:header>
```

## Example

The following is an example of a container using a header to add icons to the title bar.

```
<ui:container editable="false">
<ui:header>
<ui:link displayText="Notes">
<ui:image src="/notes.gif"/>
</ui:link>
</ui:header>
<ui:step id="c3" displayText="Container 3">
...
</ui:step>
</ui:container>
```

The following figure illustrates the preceding code.

Figure 38: Header

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

None.

## Used By

ui:container

## Content

ui:link

# Help

It displays a question mark icon with a link to a help page.

## Syntax

The following is the syntax for help.

```
<ui:help>
```

## Example

The following is an example of a container with a help icon added to the title bar.

```
<ui:container>
<ui:help displayPage="{$pages:fields_help}"/>
<ui:step id="c1" displayText="Fields" editable="true">
<ui:field-group columns="2">
...
</ui:field-group>
</ui:step>
</ui:container>
```

The following figure illustrates the preceding code. The help icon is outlined in a red box.

Figure 39: Help

## Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes for Help.

Table 32: Optional Attributes

| Attribute | Description |
|---|---|
| displayPage | URL link to location of help page. |

### Used By

ui:container

### Content

None.

## Validation

It is the field component that shows the required message with * sign for required fields.

### Syntax

The following is the syntax for validation.

```
<ui:validation>
```

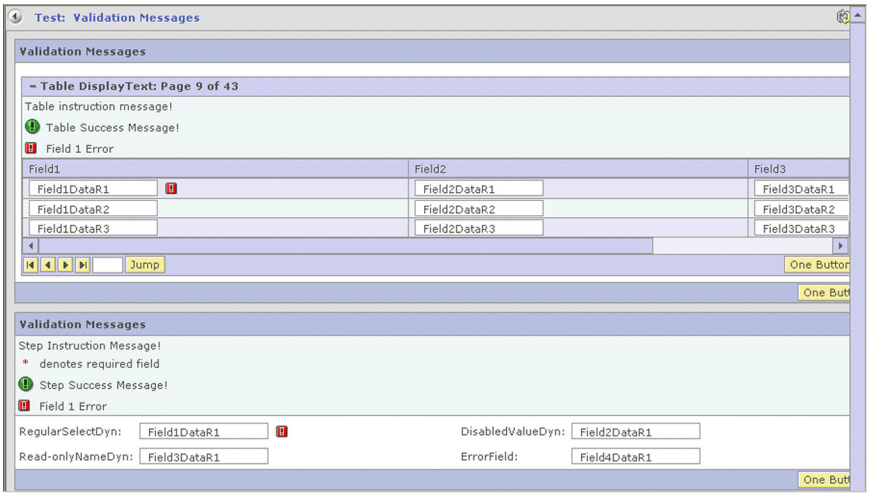
### Example

The following is an example of using UI validation.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
displayText="Validation Messages" title="Validation Messages">
  <ui:container>
    <ui:help displayPage="http://www.teradata.com"/>
   <ui:step id="c1" displayText="Validation Messages" editable="true">
      <ui:table maxRows="7" displayText="Table DisplayText"
displayTextNoRecords="Nothing found" select="REAL_DATA" rowSrc="ROW">
        <ui:validation anyFieldIsRequired="false"
instructionMessage="Table instruction message!" successMessage="Table
Success Message!" />
        <ui:field-group>
          <ui:entry-field displayText="Field1" name="FIELD_1"
editable="{FIELD_1/@Editable}" disabled="{FIELD_1/@Disabled}"/>
          <ui:entry-field displayText="Field2" name="FIELD_2"
editable="{FIELD_2/@Editable}" disabled="{FIELD_2/@Disabled}"/>
```

```
          <ui:entry-field displayText="Field3" name="FIELD_3"
editable="{FIELD_3/@Editable}" disabled="{FIELD_3/@Disabled}"/>
        </ui:field-group>
      <ui:buttons>
        <ui:button displayText="One Button2"/>
      </ui:buttons>
      </ui:table>

      <ui:buttons>
        <ui:button displayText="One Button"/>
      </ui:buttons>
    </ui:step>
  </ui:container>
  <ui:container>
      <ui:help displayPage="http://www.teradata.com"/>
    <ui:step id="c2" displayText="Validation Messages" editable="true">
        <ui:validation  instructionMessage="Step Instruction Message!"
successMessage="Step Success Message!" />
        <ui:field-group select="$root/REAL_DATA/STEP2" columns="2">
          <ui:entry-field displayText="RegularSelectDyn" name="FIELD_1"
select="FIELD_1" editable="{FIELD_1/@Editable}" disabled="{FIELD_1/
@Disabled}"/>
          <ui:entry-field displayText="DisabledValueDyn" name="FIELD_2"
value="{FIELD_2/@Value}" editable="{FIELD_2/@Editable}"
disabled="{FIELD_2/@Disabled}"/>
          <ui:entry-field displayText="Read-onlyNameDyn" name="FIELD_3"
editable="{FIELD_3/@Editable}" disabled="{FIELD_3/@Disabled}"/>
          <ui:entry-field displayText="ErrorField" name="FIELD_4"
editable="{FIELD_4/@Editable}" disabled="{FIELD_4/@Disabled}"/>
        </ui:field-group>
        <ui:buttons>
          <ui:button displayText="One Button"/>
        </ui:buttons>
      </ui:step>
  </ui:container>
</ui:page-group>
```

Figure 40: Validation Message



## Example Data

The following is an example data for preceding UI validation example:

```
<?xml version="1.0" encoding="UTF-8"?>
<RESPONSE>
  <REQ_PARAMS>
    <START_COUNT Value="57"/>
  </REQ_PARAMS>

  <HEADER>
    <APP_NAME Value="Test" />
    <HOME Value="/base/core/home.jsp" />
    <BREADCRUMBS>
      <BREADCRUMB>
        <DISPLAY_TEXT Value="Validation Messages" />
        <URL Value="/base/examples/validation/validation.x2p" />
      </BREADCRUMB>
    </BREADCRUMBS>
  </HEADER>
  <LINKS>
    <LINK Name="Return" />
    <FORMAT Date="d/MM/yyyy" />
    <CURRENT_DATE Value="8/31/2004" />
  </LINKS>

  <REAL_DATA TotalRowCount="300">
    <ROW>
      <FIELD_1 Value="Field1DataR1" Disabled="false" Editable="true">
        <_ERRORS>
          <_ERROR Value="Field 1 Error"/>
        </_ERRORS>
      </FIELD_1>
      <FIELD_2 Value="Field2DataR1" Disabled="false" Editable="true"/>
    <FIELD_3 Value="Field3DataR1" Disabled="false" Editable="true"/>
    </ROW>
    <ROW>
      <FIELD_1 Value="Field1DataR2" Disabled="false" Editable="true"/>
      <FIELD_2 Value="Field2DataR2" Disabled="false" Editable="true"/>
    <FIELD_3 Value="Field3DataR2" Disabled="false" Editable="true"/>
    </ROW>
    <ROW>
      <FIELD_1 Value="Field1DataR3" Disabled="false" Editable="true"/>
      <FIELD_2 Value="Field2DataR3" Disabled="false" Editable="true"/>
    <FIELD_3 Value="Field3DataR3" Disabled="false" Editable="true"/>
    </ROW>
    <STEP2>
      <FIELD_1 Value="Field1DataR1" Disabled="false" Editable="true">
        <_ERRORS>
          <_ERROR Value="Field 1 Error"/>
        </_ERRORS>
      </FIELD_1>
      <FIELD_2 Value="Field2DataR1" Disabled="false" Editable="true"/>
      <FIELD_3 Value="Field3DataR1" Disabled="false" Editable="true"/>
      <FIELD_4 Value="Field4DataR1" Disabled="false" Editable="true"/>
    </STEP2>
  </REAL_DATA>
</RESPONSE>
```

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for validation.

Table 33: Optional Attributes

| Attribute | Description |
|---|---|
| anyFieldHasErrors | Set to true if you wish to display default error message, ' Information is invalid. Please correct and submit again. '. Default is set to false. |
| anyFieldIsRequired | Set to false if you wish to suppress the instruction ' * denotes required field '. Default is set to true. |
| instructionMessage | Additional message to be displayed in instruction area. |
| successMessage | Alternate message to display if there are no errors. |

## Used By

ui:step, ui:table

## Content

None.

# Date

It is the field component that gives the date range.

## Syntax

The following is the syntax for date range.

```
<ui:date-range>
```

## Example

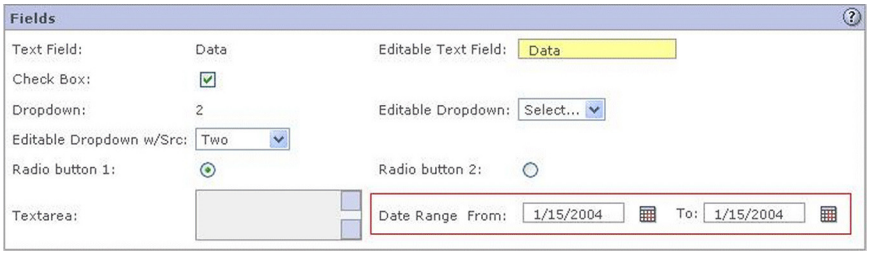
The following is an example of using a date range field.

```
<ui:field-group columns="2">
...
<ui:textarea displayText="Textarea" rows="3"/>
<ui:date-range displayText="Date Range" fromDate="1/15/2004" toDate="1/
25/2004"/>
</ui:field-group>
```

The following figure shows the date range entry field within a simple container.

**Note:** Year should be four digit number and also it should be greater than 1000. Date Picker will work only with year value of 4 digit and greater than 1000.

Figure 41: Date Range




## Example Data

Data is not required for this component.

## Required Attributes

The following are the required attributes for Date Range.

Table 34: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. |

## Optional Attributes

The following are the optional attributes for Date Range.

Table 35: Optional Attributes

| Attribute | Description |
|---|---|
| fromDate | Initial value set for the first date field. |
| toDate | Initial value set for the second date field. |
| editable | Set to true if field should be editable. Default value is true. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , 'DateTime' and 'Currency'. Default value is 'Text'. |

Table 35: Optional Attributes

| Attribute | Description |
|---|---|
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | Sets the URL for a link. Note: Attribute is deprecated. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Set to true if the header should be editable. Default is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |

Table 35: Optional Attributes

| Attribute | Description |
|---|---|
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:field-group, ui:otherwise, ui:when

### Content

ui:compare-validator, ui:links, ui:url

## Compare

### Syntax

The following is the syntax for compare.

```
<ui:compare-validator>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes for compare.

Table 36: Optional Attributes

| Attribute | Description |
|---|---|
| message | Validator message. |

Table 36: Optional Attributes

| Attribute | Description |
|---|---|
| operator | Specifies the operator of the conditional. |
| severity | Severity of the validator message. Default value is STOP. |
| withField | Specifies which named field will be used as the lefthand of the conditional. |
| withValue | Specifies the value that will be used as the righthand of the conditional. |

### Used By

None.

### Content

None.

# Page Group

It is the root level tag that defines a page or closely related pages. A page group may contain tabs and/or wizards.

### Syntax

The following is the syntax for page group.

```
<ui:page-group>
```

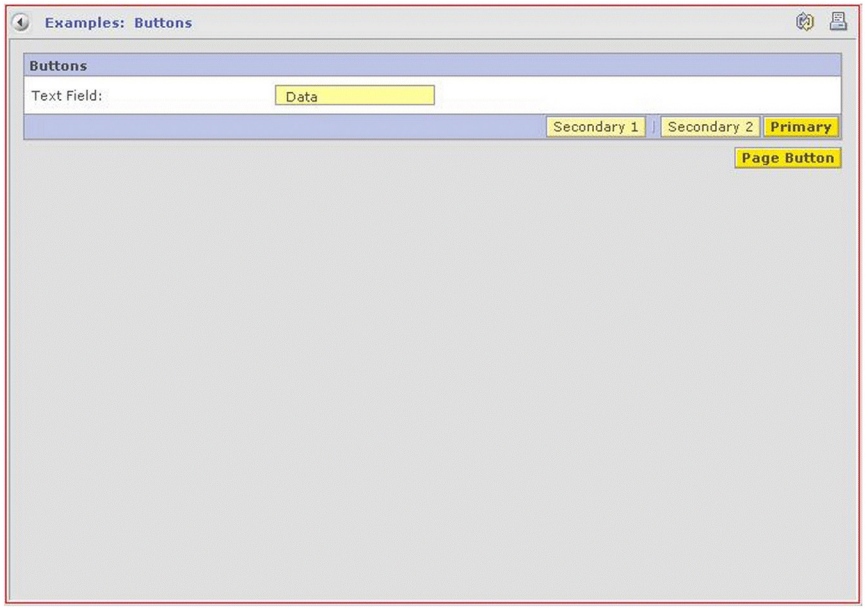
### Example

The following is an example of a page group.

```
<ui:page-group displayText="Buttons"
source="examples.buttons.buttons:load">
<ui:container>
<ui:step id="c1" displayText="Buttons" editable="true">
...
</ui:step>
</ui:container>
<ui:buttons>
<ui:button displayText="Page Button" emphasized="true"/>
</ui:buttons>
</ui:page-group>
```

The following figure shows a container with buttons inside of a page group.

Figure 42: Page Group




## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for page group.

Table 37: Optional Attributes

| Attribute | Description |
|---|---|
| columns | Number of columns on a page. |
| source | The x2 command used for retrieving the XML data. |
| title | Title of the page. Will eventually be used for specifying the breadcrumb text for this page. |
| xslfile | The xsl file to be used in lieu of configured page tags for rendering the page data. |
| showRefreshIcon | true (default value): refresh icon will be displayed and on clicking this icon, the PGL page will be refreshed.<br>false: refresh icon will be hidden and no refresh action can be done on the corresponding PGL page. |

## Used By

None.

## Content

ui:buttons, ui:container, ui:grid, ui:script, ui:search, ui:table

# Chart

It renders a chart. The first child's(chart-field) values are taken as the x-axis labels of the chart.

## Syntax

The following is the syntax for chart field.

```
<ui:chart>
```

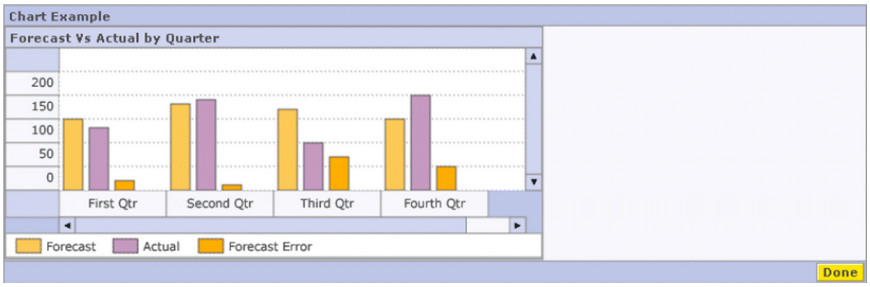
## Example

The following is an example of using the chart component.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
  <ui:container id="topContainer">
    <ui:step id="chartControl" displayText="Chart Example">
      <ui:chart width="500" height="170" xLabelsWidth="100"
yAxisNumTicks="4" displayText="Forecast Vs Actual by Quarter"
select="$root/LINE">
        <ui:chart-field displayText="Quarter" name="QUARTER"/>
        <ui:chart-field displayText="Forecast" name="FORECAST"/>
        <ui:chart-field displayText="Actual" name="ACTUAL"/>
        <ui:chart-field displayText="Forecast Error" name="ERROR"/>
      </ui:chart>
      <ui:buttons>
        <ui:button id="DONE" name="DONE" displayText="Done"
emphasized="true"/>
      </ui:buttons>
    </ui:step>
  </ui:container>
</ui:page-group>
```

The following figure shows a chart within a simple container.

Figure 43: Chart




## Example Data

Data from Example Project

```
Workflow: BasicControls
Step: Chart
PreProcessing:
<action>
  <TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <LINE>
        <ROW>
          <QUARTER Value="First Qtr"/>
          <FORECAST Value="150"/>
          <ACTUAL Value="130"/>
          <ERROR Value="20"/>
        </ROW>
        <ROW>
          <QUARTER Value="Second Qtr"/>
          <FORECAST Value="180"/>
          <ACTUAL Value="190"/>
          <ERROR Value="10"/>
        </ROW>
        <ROW>
          <QUARTER Value="Third Qtr"/>
          <FORECAST Value="170"/>
          <ACTUAL Value="100"/>
          <ERROR Value="70"/>
        </ROW>
        <ROW>
          <QUARTER Value="Fourth Qtr"/>
          <FORECAST Value="150"/>
          <ACTUAL Value="200"/>
          <ERROR Value="50"/>
        </ROW>
      </LINE>
    </SAMPLE_XML>
  </TO_DOCVAR>
</action>
```

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Chart.

Table 38: Optional Attributes

| Attribute | Description |
|---|---|
| containerId | Default is chart_container. |
| displayText | Text to be displayed as chart's title. Automatically internalized. |
| height | Height of the chart view. Default is 200. |
| select | An XPath expression used for gathering data for the field groups children. The value is automatically evaluated as an expression. |
| width | Width of the chart view. Default is 390. |
| xLablesWidth | Width of the x-axis labels. Default is 110. |
| yAxisNumTicks | Number of y-axis labels. Default is 3. |

## Used By

ui:step

## Content

ui:chart-field

# Root

It is the dummy root for intellisense.

## Syntax

The following is the syntax for Root.

```
<ui:_root>
```

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

None.

### Used By

None.

### Content

ui:page-group

# Custom Validator

### Syntax

The following is the syntax for custom validator.

```
<ui:custom-validator>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes for Custom Validator.

Table 39: Optional Attributes

| Attribute | Description |
|---|---|
| message | Validator message. |
| severity | Default is ‘stop’. |
| validationFunction | Specifies the function to use for validation. |

### Used By

None.

### Content

None.

# For Each

## Syntax

The following is the syntax for For each.

```
<ui:for-each>
```

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 40: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |

## Used By

ui:page-group, ui:step

## Content

ui:field-group, ui:table

# Login

It is the field component that shows the login page.

## Syntax

The following is the syntax for Login.

```
<ui:login>
```

## Example

```
<ui:page-group displayText="Buttons"
source="examples.buttons.buttons:load">
<ui:login  displayText="Login Page" user-name="admin"/>
</ui:page-group>
```

Figure 44 displays the preceding code.

Figure 44: Login

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 41: Optional Attributes

| Attribute | Description |
|---|---|
| background | Sets the background image to use. |
| changePwd | If set to true, a change password button is created. |
| command | Sets the X2 command that will be called on login. |

Table 41: Optional Attributes

| Attribute | Description |
|---|---|
| content-url | Sets the override URL to load instead of the home page. |
| displayPage | HTML to display to user for username/password. |
| error | Set to true if error message should be displayed. Default value is false. |
| errorDescription | Text to display if error is set to true. |
| from | Sets how the login page was accessed. For example, from the password change page or logout. |
| logo | Set the logo image to use on top left of the page. |
| panelImage | Sets the image to display on left side of page. |
| user-name | Initial username to display. |

### Used By

ui:page-group

### Content

None.

# Navigation

This component is used to construct the left navigation Links.

### Syntax

The following is the syntax for navigation.

```
<ui:navigation>
```

### Example

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
Id="Navigation.pgl">
  <ui:navigation id="aprwfl" displayText="Approval Service">
    <ui:solution id="mySolution" displayText="Solution" homePage="/base/
core/home.jsp">
      <ui:pad id="mySubmitWorkflows" name="mySubmitWorkflows"
displayText="Submit">
        <ui:pad-item id="SubmitPR" name="SubmitPR" displayText="Submit
PR" workflow="//APPROVAL/UI_Submit_PR"/>
       <ui:pad-item id="CreateNPR" name="CreateNPR" displayText="Create
NPR" workflow="//APPROVAL/UI_Submit_NPR"/>
      </ui:pad>
      <ui:pad id="myViewWorkflows" name="myViewWorkflows"
displayText="View">
```

```
        <ui:pad-item id="MyAlerts" name="MyAlerts" displayText="My
Alerts" workflow="//APPROVAL/UI_AlertInbox"/>
        <ui:pad-item id="MyFutureAlerts" name="MyFutureAlerts"
displayText="My Future Alerts" workflow="//APPROVAL/
UI_PendingAlertInbox"/>
        <ui:pad-item id="AllPendingAlerts" name="AllPendingAlerts"
displayText="All Pending Alerts" workflow="//APPROVAL/UI_Alerts"/>
        <ui:pad-item id="ApprovalHistory" name="ApprovalHistory"
displayText="Approval History" workflow="//APPROVAL/UI_ApprovalHist"/>
        <ui:pad-item id="UserList" name="UserList" displayText="User
List" workflow="//APPROVAL/UI_UserList"/>
        <ui:pad-item id="DocumentList" name="DocumentList"
displayText="Document List" workflow="//APPROVAL/UI_DocumentList"/>
      </ui:pad>
      <ui:pad id="myTestWorkflows" name="myTestWorkflows"
displayText="Test">
        <ui:pad-item id="TestMultilineNode" name="TestMultilineNode"
displayText="Test Multiline Node" workflow="//APPROVAL/
UI_TestMultiline"/>
      </ui:pad>
    </ui:solution>
  </ui:navigation>
</ui:page-group>
```

Figure 45 displays the preceding code.

Figure 45: Navigation

## Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 42: Optional Attributes

| Attribute | Description |
|---|---|
| name | Specifies the name of the parameter that is submitted as part of the form. |

### Used By

ui:page-group

### Content

ui:pad, ui:solution

## Non-Breaking Space

### Syntax

The following is the syntax for Non Breaking Space.

```
<ui:non-breaking-space>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 43: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| count | Number of non-breaking spaces to output. Default is '1'. |

Table 43: Optional Attributes

| Attribute | Description |
|---|---|
| displayCondition | Conditional that determines if the component should be displayed or not. |

## Used By

ui:links

## Content

None.

# Pad

It defines the title of the pad for navigation component.

## Syntax

The following is the syntax for Pad.

```
<ui:pad>
```

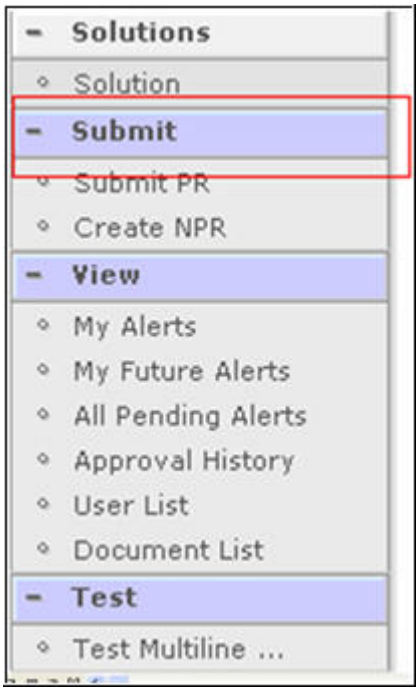
## Example

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
Id="Navigation.pgl">
  <ui:navigation id="aprwfl" displayText="Approval Service">
    <ui:solution id="mySolution" displayText="Solution" homePage="/base/
core/home.jsp">
      <ui:pad id="mySubmitWorkflows" name="mySubmitWorkflows"
displayText="Submit">
        <ui:pad-item id="SubmitPR" name="SubmitPR" displayText="Submit
PR" workflow="//APPROVAL/UI_Submit_PR"/>
       <ui:pad-item id="CreateNPR" name="CreateNPR" displayText="Create
NPR" workflow="//APPROVAL/UI_Submit_NPR"/>
      </ui:pad>
      <ui:pad id="myViewWorkflows" name="myViewWorkflows"
displayText="View">
        <ui:pad-item id="MyAlerts" name="MyAlerts" displayText="My
Alerts" workflow="//APPROVAL/UI_AlertInbox"/>
        <ui:pad-item id="MyFutureAlerts" name="MyFutureAlerts"
displayText="My Future Alerts" workflow="//APPROVAL/
UI_PendingAlertInbox"/>
        <ui:pad-item id="AllPendingAlerts" name="AllPendingAlerts"
displayText="All Pending Alerts" workflow="//APPROVAL/UI_Alerts"/>
        <ui:pad-item id="ApprovalHistory" name="ApprovalHistory"
displayText="Approval History" workflow="//APPROVAL/UI_ApprovalHist"/>
        <ui:pad-item id="UserList" name="UserList" displayText="User
List" workflow="//APPROVAL/UI_UserList"/>
        <ui:pad-item id="DocumentList" name="DocumentList"
displayText="Document List" workflow="//APPROVAL/UI_DocumentList"/>
```

```
      </ui:pad>
      <ui:pad id="myTestWorkflows" name="myTestWorkflows"
displayText="Test">
        <ui:pad-item id="TestMultilineNode" name="TestMultilineNode"
displayText="Test Multiline Node" workflow="//APPROVAL/
UI_TestMultiline"/>
      </ui:pad>
    </ui:solution>
  </ui:navigation>
</ui:page-group>
```

Figure 46 displays the preceding code.

Figure 46: Pad




## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes.

Table 44: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |

### Used By

ui:navigation

### Content

ui:pad-item

# Pad Item

It defines the name of the link for a pad.

### Syntax

The following is the syntax for Pad Item

```
<ui:pad-item>
```

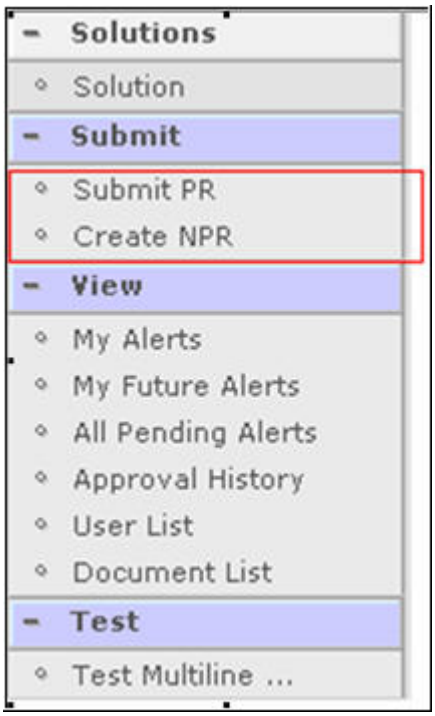
### Example

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
Id="Navigation.pgl">
  <ui:navigation id="aprwfl" displayText="Approval Service">
    <ui:solution id="mySolution" displayText="Solution" homePage="/base/
core/home.jsp">
      <ui:pad id="mySubmitWorkflows" name="mySubmitWorkflows"
displayText="Submit">
        <ui:pad-item id="SubmitPR" name="SubmitPR" displayText="Submit
PR" workflow="//APPROVAL/UI_Submit_PR"/>
       <ui:pad-item id="CreateNPR" name="CreateNPR" displayText="Create
NPR" workflow="//APPROVAL/UI_Submit_NPR"/>
      </ui:pad>
      <ui:pad id="myViewWorkflows" name="myViewWorkflows"
displayText="View">
        <ui:pad-item id="MyAlerts" name="MyAlerts" displayText="My
Alerts" workflow="//APPROVAL/UI_AlertInbox"/>
        <ui:pad-item id="MyFutureAlerts" name="MyFutureAlerts"
displayText="My Future Alerts" workflow="//APPROVAL/
UI_PendingAlertInbox"/>
        <ui:pad-item id="AllPendingAlerts" name="AllPendingAlerts"
displayText="All Pending Alerts" workflow="//APPROVAL/UI_Alerts"/>
```

```
        <ui:pad-item id="ApprovalHistory" name="ApprovalHistory"
displayText="Approval History" workflow="//APPROVAL/UI_ApprovalHist"/>
        <ui:pad-item id="UserList" name="UserList" displayText="User
List" workflow="//APPROVAL/UI_UserList"/>
        <ui:pad-item id="DocumentList" name="DocumentList"
displayText="Document List" workflow="//APPROVAL/UI_DocumentList"/>
      </ui:pad>
      <ui:pad id="myTestWorkflows" name="myTestWorkflows"
displayText="Test">
        <ui:pad-item id="TestMultilineNode" name="TestMultilineNode"
displayText="Test Multiline Node" workflow="//APPROVAL/
UI_TestMultiline"/>
      </ui:pad>
    </ui:solution>
  </ui:navigation>
</ui:page-group>
```

Figure 47 displays the preceding code.

Figure 47: Pad Item




## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 45: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |
| name | Specify the name of the parameter that is submitted as part of the form |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| type | Type of pad display. Options are 'solution' and 'application. Default is 'application'. |
| url | Set the URL value for a link. |

### Used By

ui:pad, ui:pad-item

### Content

ui:pad-item

# Sort

### Syntax

The following is the syntax for sort.

```
<ui:sort>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 46: Optional Attributes

| Attribute | Description |
|---|---|
| name | Default is 'SORT.' |

## Used By

None.

## Content

None.

# Image

It is used to display an image.

## Syntax

The following is the syntax for Image.

```
<ui:image>
```

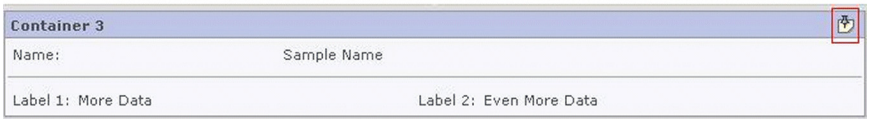
## Example

The following is an example of adding an image, used as an icon, to the title bar.

```
<ui:container editable="false">
<ui:header>
<ui:link displayText="Notes">
<ui:url page="{$pages:note_details}"/>
<ui:image src="/notes.gif"/>
</ui:link>
</ui:header>
<ui:step id="c3" displayText="Container 3">
...
</ui:step>
</ui:container>
```

The following figure illustrates the preceding code.

Figure 48: Image



## Example Data

### *Data from Example Project*

**Workflow**: BasicControls

Step: Image

**PGL**: Data can be specified directly using the src attribute

```
<ui:image name="TD_MAST" displayText="Teradata Masthead Image" src="/
masthead.gif" height="25" width="500"/>
```

Alternatively src can be set in the data as follows

PGL:

```
<ui:image name="TD_MAST" displayText="Teradata Masthead Image"
src="{concat('/',$root/Image/@Value)}" height="25" width="500" />
```

**Data:**

```
<action>
      <TO_DOCVAR AssignToVar="inXml">
         <SAMPLE_XML>
            <Image Value="masthead.gif"/>
         </SAMPLE_XML>
      </TO_DOCVAR>
</action>
```

Figure 49: Image Example

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Image.

Table 47: Optional Attributes

| Attribute | Description |
|---|---|
| src | URL link to the location of image. |

## Used By

ui:link

### Content

None.

# Image Link Field

Displays an image that has been hyperlinked. Can be used within a field group.

### Syntax

The following is the syntax for image link field.

```
<ui:image-link-field>
```

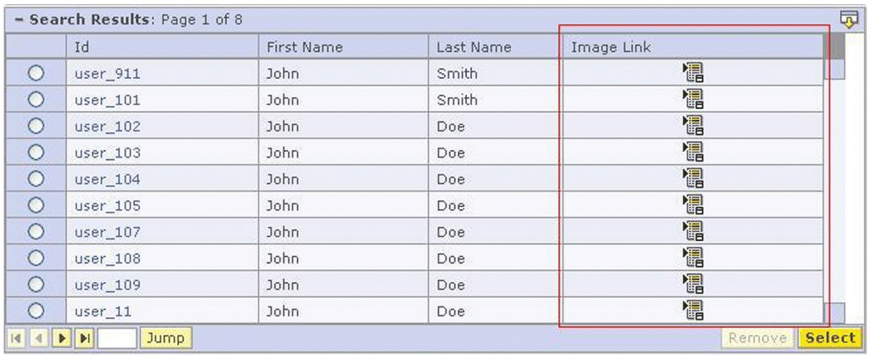
### Example

The following is an example of a column of link fields in a simple table.

```
<ui:table displayText="Search Results" displayTextNoRecords="No Users
found" select="USER_PROFILES" selectable="Single">
<ui:field-group>
<ui:link-field displayText="Id">
<ui:url page="{$pages:user_profile}">
  <ID select="ID/@Value"/>
</ui:url>
</ui:link-field>
<ui:display-field displayText="First Name" name="FIRST_NAME"/>
<ui:display-field displayText="Last Name" name="LAST_NAME"/>
<ui:image-link-field displayText="Image Link" src="/view_details.gif"
name="ID" dataType="Constant">
<ui:url page="{$pages:user_profile}">
  <USER_ID select="ID/@Value"/>
</ui:url>
</ui:image-link-field>
</ui:field-group>
<ui:buttons>
<ui:button displayText="Remove" disabled="true"
enableBySelections="true"/>
<ui:button displayText="Select" emphasized="yes"
enableBySelections="true"/>
</ui:buttons>
</ui:table>
```

The following figure illustrates the preceding code.

Figure 50: Image Link Field




## Example Data

### *Data from Example Project*

**Workflow**: BasicControls

**Step**: Image

**PGL**: Data can be specified directly using the src attribute

```
<ui:image-link-field name="TD_LINK" displayText="Goto Teradata Homepage
(Click on the Info icon)" src="/information_sml.gif">
<ui:url page="http://www.teradata.com"/>
</ui:image-link-field>
```

## Required Attributes

The following are the required attributes for Image Link Field.

Table 48: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form |

## Optional Attributes

The following are the optional attributes for Image Link Field.

Table 49: Optional Attributes

| Attribute | Description |
|---|---|
| src | URL link to the location of image. |
| targetType | Specify the type of frame or window to direct the referenced document. The possible value is 'popup'. |

Table 49: Optional Attributes

| Attribute | Description |
|---|---|
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| dataType | Number of fraction digits to be displayed. |
| decimals | Default value used if no value is initially given. |
| defaultValue | Conditional that determines if the field or column should be displayed or not. |
| displayCondition | Set to true if no wrapping is desired within the field. |
| displayNoWrap | Set to true if no wrapping is desired within the field. Default value is true. |
| displayPage | Text to be displayed as the label or header of the field. Automatically internationalized. |
| displayText | Possible value is T_FIELD_ENTRY. |
| elementName | Format to use other than the default. |
| format | Specify a style class controlling the appearance of header cell's contents. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |
| headerClass | Specifies a style class controlling the appearance of the header cell’s contents. |
| headerEditable | Set to true if the header should be editable. Default value is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. Note: Attribute is deprecated. |
| headerLink | Sets the URL for the header. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |

Table 49: Optional Attributes

| Attribute | Description |
|---|---|
| rowSpan | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:field-group, ui:otherwise, ui:when

### Content

ui:url, ui:compare-validator, ui:links, ui:url

# Link

A link displays an icon on a header.

### Syntax

The following is the syntax for link:

```
<ui:link>
```

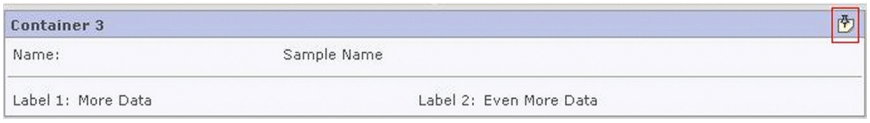
## Example

The following is an example of using a icon Link for adding special actions to a container title bar.

```
<ui:container editable="false">
<ui:header>
<ui:link displayText="Notes">
<ui:url page="{$pages:note_details}"/>
<ui:image src="/notes.gif"/>
</ui:link>
</ui:header>
<ui:step id="c3" displayText="Container 3">
...
</ui:step>
</ui:container>
```

The following figure shows a notes detail icon that is added to the container title bar.

Figure 51: Link




## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for a link.

Table 50: Optional Attributes

| Attribute | Description |
|---|---|
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| onClick | Specify the javascript to execute if the text or image is clicked. |
| targetName | Define the name of the frame or window to receive the referenced document. |
| targetType | Specify the type of frame or window to direct the referenced document. The possible value is 'popup'. |

### Used By

ui:header, ui:links

### Content

ui:image, ui:url

# Link Field

Link field displays text that has been hyperlinked. It is similar to link but it can be used within a field group.

### Syntax

The following is the syntax for Link Field.

```
<ui:link-field>
```

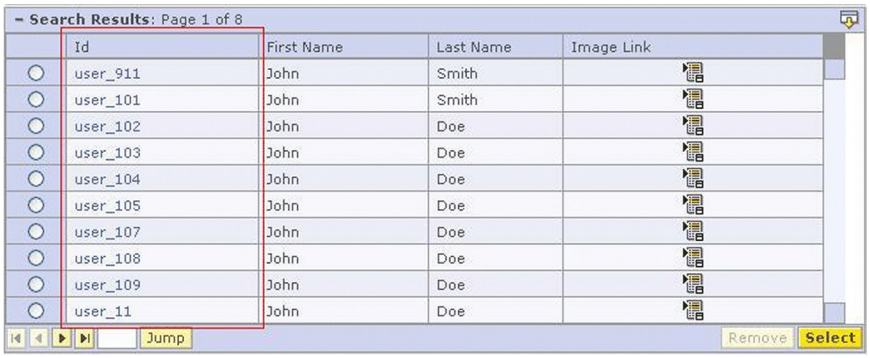
### Example

The following is an example of using a column of link fields in a simple table.

```
<ui:table displayText="Search Results" displayTextNoRecords="No Users
found" select="USER_PROFILES" selectable="Single">
<ui:field-group>
<ui:link-field displayText="Id">
<ui:url page="{$pages:user_profile}">
   <ID select="ID/@Value"/>
</ui:url>
</ui:link-field>
<ui:display-field displayText="First Name" name="FIRST_NAME"/>
<ui:display-field displayText="Last Name" name="LAST_NAME"/>
<ui:image-link-field displayText="Image Link" src="/view_details.gif"
name="ID" dataType="Constant">
<ui:url page="{$pages:user_profile}">
   <USER_ID select="ID/@Value"/>
</ui:url>
</ui:image-link-field>
</ui:field-group>
<ui:buttons>
<ui:button displayText="Remove" disabled="true"
enableBySelections="true"/>
<ui:button displayText="Select" emphasized="yes"
enableBySelections="true"/>
</ui:buttons>
</ui:table>
```

The following figure illustrates the preceding code.

Figure 52: Link Field




## Example Data

### *Data from Example Project*

Data from Example Project

**Workflow**: BasicControls

**Step**: Basic

**PGL:**

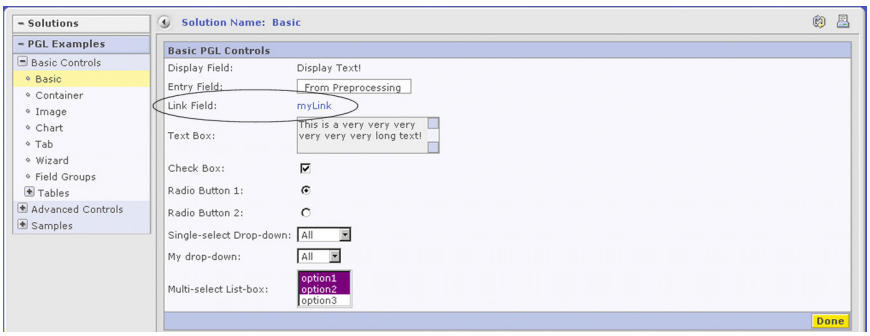
```
<ui:link-field name="LINK_FIELD" displayText="Link Field"
value="Teradata Corporation" id="LINK_FIELD">
<ui:url page="http://www.teradata.com">
<BUTTON_ID select="'TD_HOME'"/>
</ui:url>
</ui:link-field>
```

Alternatively value can be specified as value="{$root/LinkFieldValue/@Value}" and preprocessing will have

**PreProcessing:**

```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
            <LinkFieldValue Value="Teradata Corporation"/>
     </SAMPLE_XML>
</TO_DOCVAR>
```

Figure 53: Link



## Required Attributes

The following are the required attributes for Link Field.

Table 51: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form |

## Optional Attributes

The following are the optional attributes for Link Field.

Table 52: Optional Attributes

| Attribute | Description |
|---|---|
| targetName | Define the name of the frame or window to receive the referenced document. |
| targetType | Specify the type of frame or window to direct the referenced document. The possible value is 'popup'. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’, and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | URL link value which might refer to pages.cnd definition or user defined http URL. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |

Table 52: Optional Attributes

| Attribute | Description |
|---|---|
| headerEditable | Set to “true” if header should be editable. Default is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |
| filterType | Specifies the type of Filter field, filter must be set to true. Possible Types ‘Text’, ‘Dropdown’ |
| filterValuesSelect | An Xpath expression used for populating the filter values, if filter type is Dropdown. |

### Used By

ui:field-group, ui:otherwise, ui:when

### Content

ui:compare-validator, ui:links, ui:url,

## Links

Currently unreachable.

### Syntax

The following is the syntax for Links.

```
<ui:links>
```

### Example

None.

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

None.

### Used By

ui:link-field

### Content

ui:button, ui:link, ui:non-breaking-space

## URL

It adds a url link to an icon.

## Syntax

The following is the syntax for URL.

```
<ui:url>
```

## Example

The following is an example of using an icon link to add special actions to a container title bar.

```
<ui:container editable="false">
<ui:header>
<ui:link displayText="Notes">
<ui:url page="{$pages:note_details}"/>
<ui:image src="/notes.gif"/>
</ui:link>
</ui:header>
<ui:step id="c3" displayText="Container 3">
...
</ui:step>
</ui:container>
```

The following figure illustrates the preceding code.

Figure 54: URL

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for URL.

Table 53: Optional Attributes

| Attribute | Description |
|---|---|
| Page | The URL value for the component. |

## Used By

ui:image-link-field, ui:link

## Content

None.

# UI:DIV

The <UI:DIV> tag defines a division or a section in an HTML document. The <UI:DIV> tag is used to group block-elements to format them with styles.

## Syntax

The following is the syntax for <UI:DIV> tag.

```
<ui:div id="ID_FOR_DIV_TAG" style="width: 800px; height: 400px;"/>
```

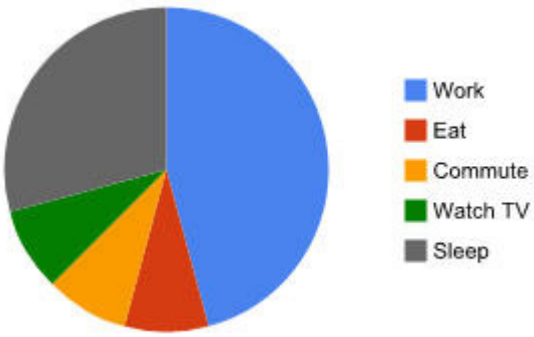
## Example

The following is an example of using <UI:DIV> tag.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
  <ui:container id="container">
    <ui:step id="top_container" displayText="Summary Screen">
      <ui:script id="script1" src="http://www.google.com/jsapi" />
      <ui:script id="script2">google.load('visualization', '1',
{packages: ['piechart']});
            function drawVisualization()
        {
               // Create and populate the data table.
               var data = new google.visualization.DataTable();
               data.addColumn('string', 'Task');
               data.addColumn('number', 'Hours per Day');
               data.addRows(5);
               data.setValue(0, 0, 'Work');
               data.setValue(0, 1, 11);
               data.setValue(1, 0, 'Eat');
               data.setValue(1, 1, 2);
               data.setValue(2, 0, 'Commute');
               data.setValue(2, 1, 2);
               data.setValue(3, 0, 'Watch TV');
               data.setValue(3, 1, 2);
               data.setValue(4, 0, 'Sleep');
               data.setValue(4, 1, 7);

               // Create and draw the visualization.
               new
google.visualization.PieChart(document.getElementById('visualization')).
               draw(data, null);
            }
            google.setOnLoadCallback(drawVisualization);</ui:script>
      <ui:div id="visualization" style="width: 300px; height: 300px;" />
    </ui:step>
  </ui:container>
</ui:page-group>
```

The following figure illustrates the preceding code.

Figure 55: UI:DIV




## Example Data

Data is not required for this component.

## Required Attributes

The following table lists the required attributes for UI:DIV tag.

Table 54: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |

## Optional Attributes

The following table lists the optional attributes for UI:DIV tag.

Table 55: Optional Attributes

| Attribute | Description |
|---|---|
| Style | Specify a style controlling the appearance of cell's contents. |

## Used By

Step tag

## Content

None.

# Table

The table is used to render tabular data in the Teradata standard style.

## Syntax

The following is the syntax for a table:

```
<ui:table>
```

## Examples

The following are the examples of using a table.

Example 1:

The following is an example of a simple table populated from a search container.

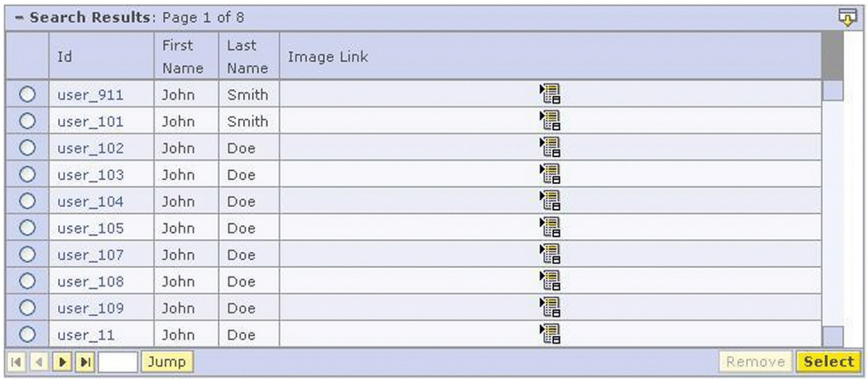
```
<ui:table displayText="Search Results" displayTextNoRecords="No Users
found" select="USER_PROFILES" selectable="Single" postRowIdAs="USER_ID">
      <ui:field-group>
         <ui:link-field displayText="Id">
            <ui:url page="{$pages:user_profile}">
             <ID select="ID/@Value"/>
            </ui:url>
         </ui:link-field>
         <ui:display-field displayText="First Name" name="FIRST_NAME"/>
         <ui:display-field displayText="Last Name" name="LAST_NAME"/>
         <ui:image-link-field displayText="Image Link" src="/
         view_details.gif" name="ID" dataType="Constant">
         <ui:url page="{$pages:user_profile}">
          <USER_ID select="ID/@Value"/>
         </ui:url>
         </ui:image-link-field>
      </ui:field-group>
      <ui:buttons>
         <ui:button displayText="Remove" disabled="true"
         enableBySelections="true"/>
         <ui:button displayText="Select" emphasized="yes"
         enableBySelections="true"/>
      </ui:buttons>
</ui:table>
```

The following is an example of the input variable for the above table:

```
<ROOT>
      <USER_PROFILES>
         <USER_PROFILE>
            <ID Value="user_911"/>
            <FIRST_NAME Value="John"/>
            <LAST_NAME Value="Smith"/>
            <USER_ID Value="user_911"/>
         </USER_PROFILE>
         <USER_PROFILE>
            <ID Value="user_101"/>
            <FIRST_NAME Value="John"/>
            <LAST_NAME Value="Smith"/>
            <USER_ID Value="user_101"/>
         </USER_PROFILE>
      </USER_PROFILES>
</ROOT>
```

The following figure illustrates the preceding code.

Figure 56: Simple Table




The following is an example of the output variable that holds the id for the row selected by the user

```
<PARAMETERS>
      <USER_ID Value="user_911"/>
</PARAMETERS>
```

The postRowIdAs attribute was set to “ID” in the table definition. Hence, the USER_ID field was output as the id for the selected row.

The following is an example of the output variable if user selected multiple rows (this requires selectable="All"):

```
<PARAMETERS>
      <USER_ID Value="user_911"/>
      <USER_ID Value="user_101"/>
</PARAMETERS>
```

## Data from Example Project

**Workflow**: BasicControls

**Step**: Simple

**PGL:**

```
 <ui:table id="tableExample1" displayText="Table (without Paging)"
rowSrc="$root/LINE" selectable="All" postRowIdAs="SELECTED_ITEM"
rowId="ITEM" maxRows="-1" scrollable="false">
    <ui:field-group id="tableColumns">
      <ui:display-field name="ITEM" displayText="Item"/>
      <ui:display-field name="DESC" displayText="Description"/>
    </ui:field-group>
  </ui:table>
```

**PreProcessing:**

```
<action>
  <TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <LINE>
        <ITEM Value="Item-001"/>
        <DESC Value="Item Code 001"/>
      </LINE>
```

```
      <LINE>
        <ITEM Value="Item-002"/>
        <DESC Value="Item Code 002"/>
      </LINE>
      <LINE>
        <ITEM Value="Item-003"/>
        <DESC Value="Item Code 003"/>
      </LINE>
      <TOTAL_ROWS Value="4"/>
    </SAMPLE_XML>
  </TO_DOCVAR>
</action>
```

Example 2:

The following are examples of using the Pagination functionality in tables:

```
<ui:table id="bottomTable" displayText="Store" select="STORE"
selectable="Single" postRowIdAs="SELECTED_STORE" rowId="StoreId"
scrollable="true" maxRows="-1">
```

**Note:** maxRows=”-1” will retrieve all rows and maxRows=”10” will retrieve 10 rows per page.

The following are the two possible cases for pagination using the doPaging attribute:

1 doPaging=”false” maxRows=”-1”

Use the above statement if the server is rendering the page. Use $thisParam/START_ROW to get the startRow and use ReturnRowCount=”yes” on GET_DOCUMENT to get the number of rows returned by the server.

For example,

```
<ui:table doPaging="true" rowId="ItemID" collapsable="false"
postRowIdAs="selected_item" displayText="Search Results"
displayTextNoRecords="No Data found" select="ItemList"
selectable="Single" scrollable="true" totalRowCount="$root/
TotalRowCount/@Value" startAtRow="$root/startRow/@Value"
maxRows="10">
<SET Var="startRow" FromSelect="$thisParam/START_COUNT/@Value"/>
<GET_DOCUMENT AssignToVar="Items" ServiceName="BCM_MASTER"
Name="ItemMaster" ReturnRowCount="yes" StartAtRow="{$startRow}">
    <Status Value="{$status}" MatchBy="EQUAL"/>
</GET_DOCUMENT>
<ROOT>
    <ItemList TotalRowCount="{$Items/@TotalRowCount}"  />
</ROOT>
```

2 doPaging =”true”

Use the above statement if renderer is doing the pagination

For example,

When dealing with data that has not been stored to the server yet (so you cannot do GET_DOCUMENT on it). For example, temporary data in workflow context. This is not efficient in case of large number of rows in a table.

For example,

```
<ROOT>
    <ItemList TotalRowCount="50”/>
```

```
</ROOT>
```

## Example Data

### *Data from Example Project*

**Workflow**: BasicControls

**Step**: Simple

**PGL:**

```
 <ui:table id="tableExample2" displayText="Table (with Paging)"
rowSrc="$root/LINE" selectable="Single" postRowIdAs="SELECTED_ITEM"
rowId="ITEM" maxRows="3" totalRowCount="$root/TOTAL_ROWS/@Value"
exportable="true" scrollable="false">
    <ui:field-group id="tableColumns">
      <ui:display-field name="ITEM" displayText="Item"/>
      <ui:display-field name="DESC" displayText="Description"/>
    </ui:field-group>
  </ui:table>
```

Preprocessing:

Same data as in Example 2 above.

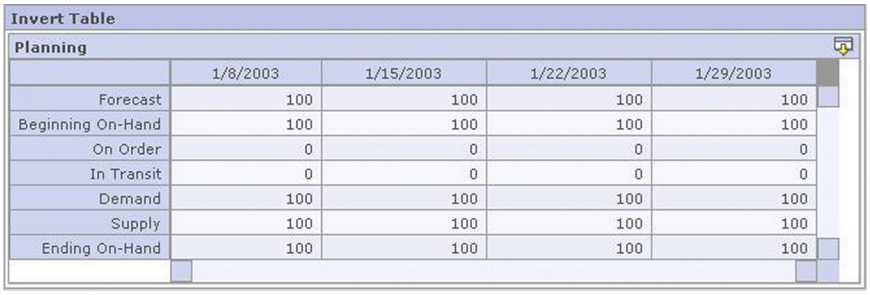
Example 3:

The following is an example of a table with a treecell column.

```
<ui:table displayText="Orders" displayTextNoRecords="No Orders"
select="CUSTOMER_ORDERS" rowSubTreeSrc="ORDER_LINE_ITEMS/
ORDER_LINE_ITEM" collapsable="false">
<ui:field-group>
<ui:treecell displayText="Order ID" name="ID"
noChildren="{count(ORDER_LINE_ITEMS/ORDER_LINE_ITEM) = 0}"
depth="{count(ancestor::node()[name()='ORDER_LINE_ITEMS'])}" column="0">
<ui:url page="{$pages:standard_order_select_order_view}">
  <ID select="ID/@Value"/>
</ui:url>
</ui:treecell>
<ui:display-field displayText="Customer Name" name="CUST_ORG_NAME"/>
<ui:display-field displayText="Status" name="STATE_ID"/>
<ui:display-field displayText="Item Id" name="ITEM_ID"/>
<ui:display-field displayText="Request Date" name="DELIVERY_DATE"
dataType="Date"/>
<ui:display-field displayText="Request Quantity" name="REQ_QTY"/>
<ui:display-field displayText="Ship Date" name="SHIP_DATE"
dataType="Date"/>
<ui:display-field displayText="Promise Status" name="PROMISE_STATE"/>
</ui:field-group>
</ui:table>
```

The following figure illustrates the preceding code.

Figure 57: Inverted Table




| | 1/8/2003 | 1/15/2003 | 1/22/2003 | 1/29/2003 |
|---|---|---|---|---|
| Forecast | 100 | 100 | 100 | 100 |
| Beginning On-Hand | 100 | 100 | 100 | 100 |
| On Order | 0 | 0 | 0 | 0 |
| In Transit | 0 | 0 | 0 | 0 |
| Demand | 100 | 100 | 100 | 100 |
| Supply | 100 | 100 | 100 | 100 |
| Ending On-Hand | 100 | 100 | 100 | 100 |

### *Data from Example Project*

**Workflow**: BasicControls

**Step**: Tree Cell

**PGL:**

```
<ui:table displayText="Table with TreeCell" rowSrc="$root/
CUSTOMER_ORDERS/CUSTOMER_ORDER" rowSubTreeSrc="ORDER_LINE_ITEMS/
ORDER_LINE_ITEM" collapsable="false" maxRows="-1">
<ui:field-group>
        <ui:treecell displayText="Order ID" name="ID"
noChildren="{count(ORDER_LINE_ITEMS/ORDER_LINE_ITEM) = 0}"
depth="{count(ancestor::node()[name()='ORDER_LINE_ITEMS'] )}"
column="0"/>
        <ui:display-field displayText="Customer Name" name="CUSTOMER"/>
        <ui:display-field displayText="Status" name="STATUS"/>
        <ui:display-field displayText="Item" name="ITEM"/>
        <ui:display-field displayText="Request Date" name="REQ_DATE"
dataType="Date"/>
        <ui:display-field displayText="Request Quantity" name="QTY"/>
        <ui:display-field displayText="Ship Date" name="SHIP_DATE"
dataType="Date"/>
      </ui:field-group>
       </ui:table>
```

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
      <SAMPLE_XML>
        <CUSTOMER_ORDERS>
          <CUSTOMER_ORDER>
            <ID Value="CO-1"/>
            <CUSTOMER Value="Teradata Corporation"/>
            <STATUS Value="Open"/>
            <ORDER_LINE_ITEMS>
              <ORDER_LINE_ITEM>
                <ITEM Value="Item-01"/>
                <REQ_DATE Value="{date()}"/>
                <QTY Value="10"/>
                <SHIP_DATE Value="{date()}"/>
              </ORDER_LINE_ITEM>
              <ORDER_LINE_ITEM>
                <ITEM Value="Item-02"/>
                <REQ_DATE Value="{date()}"/>
                <QTY Value="20"/>
```

```
                    <SHIP_DATE Value="{date()}"/>
                  </ORDER_LINE_ITEM>
                </ORDER_LINE_ITEMS>
              </CUSTOMER_ORDER>
            </CUSTOMER_ORDERS>
          </SAMPLE_XML>
        </TO_DOCVAR>
```

Figure 58: Tree Cell Example

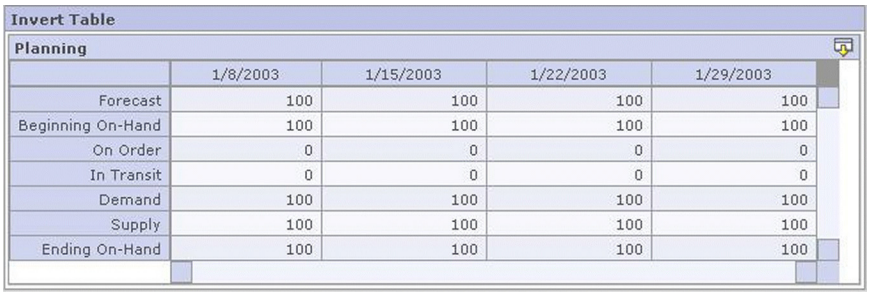
Example 4:

The following is an example of an inverted table that is a table with the rows and columns inverted.

```
<ui:table invert="true" displayText="Planning" displayTextNoRecords="No
Programs found" select="TPAS" rowSrc="TPA" collapsable="false"
maxRows="-1" selectable="None">
      <ui:help url="{$pages:search_results_help}"/>
      <ui:field-group>
         <ui:display-field displayText="" name="FROM_DATE"
         dataType="Date" align="center"/>
         <ui:display-field displayText="Forecast" name="FORECAST"
         dataType="Number"/>
         <ui:display-field displayText="Beginning On-Hand" name="BOH"
         dataType="Number"/>
         <ui:display-field displayText="On Order" name="ON_ORDER"
         dataType="Number"/>
         <ui:display-field displayText="In Transit" name="IN_TRANSIT"
         dataType="Number"/>
         <ui:display-field displayText="Demand" name="DEMAND"
         dataType="Number"/>
         <ui:display-field displayText="Supply" name="SUPPLY"
         dataType="Number"/>
         <ui:display-field displayText="Ending On-Hand" name="EOH"
         dataType="Number"/>
      </ui:field-group>
</ui:table>
```

The following figure illustrates the preceding code.

Figure 59: Inverted Table




| | 1/8/2003 | 1/15/2003 | 1/22/2003 | 1/29/2003 |
|---|---|---|---|---|
| Forecast | 100 | 100 | 100 | 100 |
| Beginning On-Hand | 100 | 100 | 100 | 100 |
| On Order | 0 | 0 | 0 | 0 |
| In Transit | 0 | 0 | 0 | 0 |
| Demand | 100 | 100 | 100 | 100 |
| Supply | 100 | 100 | 100 | 100 |
| Ending On-Hand | 100 | 100 | 100 | 100 |

### *Data from Example Project*

**Workflow**: BasicControls

**Step**: Simple

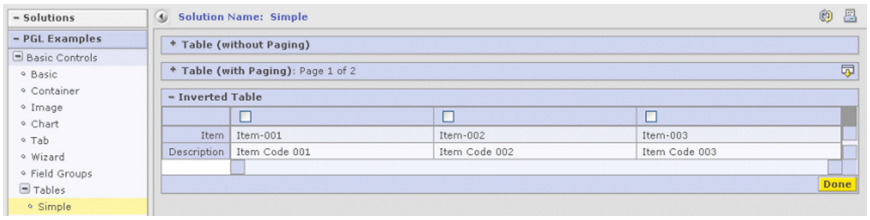
**PGL:**

```
<ui:table id="tableExample3" displayText="Inverted Table" rowSrc="$root/
LINE" selectable="All" postRowIdAs="SELECTED_ITEM" rowId="ITEM"
invert="true" maxRows="-1">
      <ui:field-group id="tableColumns">
        <ui:display-field name="ITEM" displayText="Item"/>
        <ui:display-field name="DESC" displayText="Description"/>
      </ui:field-group>
</ui:table>
```

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <LINE>
            <ITEM Value="Item-001"/>
            <DESC Value="Item Code 001"/>
          </LINE>
          <LINE>
            <ITEM Value="Item-002"/>
            <DESC Value="Item Code 002"/>
          </LINE>
          <LINE>
            <ITEM Value="Item-003"/>
            <DESC Value="Item Code 003"/>
          </LINE>
            <TOTAL_ROWS Value="4"/>
        </SAMPLE_XML>
</TO_DOCVAR>
```

Figure 60: Table Example

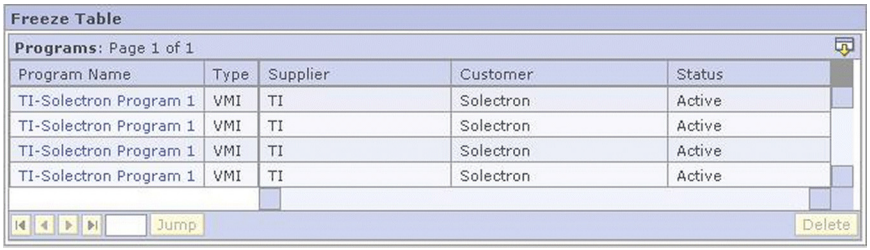


Example 5:

The following is an example of a freeze table with two column frozen.

```
<ui:table freeze="2" displayText="Programs" displayTextNoRecords="No
Programs found" select="TPAS" rowSrc="TPA" collapsable="false"
selectable="None">
<ui:help url="{$pages:search_results_help}"/>
<ui:field-group>
<ui:link-field name="TPA_NAME" displayText="Program Name">
<ui:url page="{$pages:cpm_program_review}">
  <TPA_ID select="TPA_ID/@Value"/>
</ui:url>
</ui:link-field>
<ui:display-field displayText="Type" name="TPA_TYPE"/>
<ui:display-field displayText="Supplier" name="SUPP_NAME"/>
<ui:display-field displayText="Customer" name="CUST_NAME"/>
<ui:display-field displayText="Status" name="STATUS"/>
</ui:field-group>
<ui:pagination/>
<ui:buttons>
<ui:button displayText="Delete" disabled="true"/>
</ui:buttons>
</ui:table>
The following figure illustrates the preceding code.
```

Figure 61: Columns Frozen in the Table




Example 6:

The following is an example of a complex table with multi-leveled field groups. One of the columns is using the choose-field to show heterogeneous fields within a single column.

```
<ui:table displayText="Programs" scrollable="false"
displayTextNoRecords="No Programs found" select="TPAS" rowSrc="TPA"
collapsable="false" selectable="Single">
<ui:help url="{$pages:search_results_help}"/>
<ui:field-group>
<ui:link-field name="TPA_NAME" displayText="Program Name">
<ui:url page="{$pages:cpm_program_review}">
  <TPA_ID select="TPA_ID/@Value"/>
</ui:url>
</ui:link-field>
<ui:choose-field name="EVENT_ID" displayText="Choose Field">
<ui:when test="TPA_TYPE/@Value = 'VMI'">
  <ui:display-field displayText="" name="TPA_TYPE"/>
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'CMI-ER'">
  <ui:entry-field displayText="" name="TPA_TYPE"
  value="{TPA_TYPE/@Value}"/>
```

```
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'SMI'">
  <ui:dropdown displayText="Current Forecast State"
  name="FCST_STATE" selectAll="true"
  dataType="Constant">
    <ui:option id="VALIDATION_FAILED"
    value="VALIDATION_FAILED"/>
    <ui:option id="MODIFIED" value="MODIFIED"/>
    <ui:option id="RELEASE_EXTRACTION_FAILED"
    value="RELEASE_EXTRACTION_FAILED"/>
  </ui:dropdown>
</ui:when>
<ui:otherwise>
  <ui:display-field displayText="" name="TPA_TYPE"/>
</ui:otherwise>
</ui:choose-field>
<ui:field-group displayText="Common Header">
<ui:display-field displayText="Type" name="TPA_TYPE"/>
<ui:display-field displayText="Supplier" name="SUPP_NAME"/>
<ui:display-field displayText="Customer" name="CUST_NAME"/>
</ui:field-group>
<ui:display-field displayText="Status" name="STATUS" style="{STATUS/
@Style}"/>
</ui:field-group>
<ui:pagination/>
<ui:buttons>
<ui:button displayText="Delete" enabledBySelections="true"/>
</ui:buttons>
</ui:table>
```

### *Composite Primary Key in UI: Table*

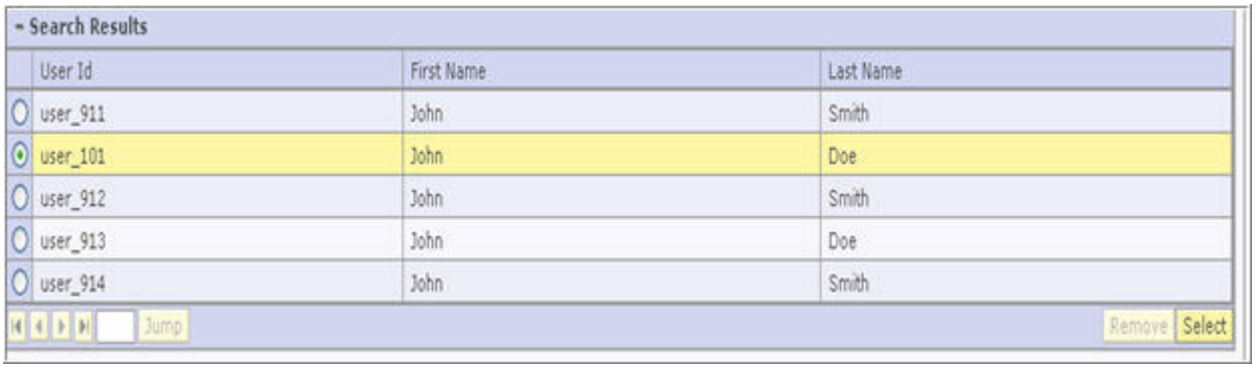
Ui: table component in PGL support only one primary key column in rowId tag for identifying the row in the table. If table has more then one primary key (composite primary key) then it fails to return the selected row. Support for composite primary key gives the user to select multiple column values.

There will be no change in the look and feel and functionality of the UI.

Example:

```
<ui:table displayText="Search Results" displayTextNoRecords="No Users
found" select="USER_PROFILES" selectable="Single"
rowId="USER_ID#FIRST_NAME#LAST_NAME" postRowIdAs="SELECTED_USER">
        <ui:field-group>
          <ui:display-field displayText="User Id" name="USER_ID"/>
          <ui:display-field displayText="First Name" name="FIRST_NAME"/>
          <ui:display-field displayText="Last Name" name="LAST_NAME"/>
        </ui:field-group>
        <ui:buttons>
          <ui:button displayText="Remove" disabled="true"
enableBySelections="true"/>
          <ui:button displayText="Select" emphasized="yes"
enableBySelections="true"/>
        </ui:buttons>
      </ui:table>
```

The below figure displays the use of composite primary key in UI:Table.




The following is an example of the output variable that holds the USER_ID, FIRST_NAME, LAST_NAME for the row selected by the user

```
<PARAMETERS>
<SELECTED_USER
Value="USER_ID=user_101#FIRST_NAME=John#LAST_NAME=Doe"/>
</PARAMETERS>
```

The postRowIdAs attribute was set to "SELECTED_USER" in the table definition. Hence, the SELECTED_USER field was output with the values of the column present in the rowId="USER_ID#FIRST_NAME#LAST_NAME".

The following is an example of the output variable if user selected multiple rows (this requires selectable="All"):

```
<PARAMETERS>
<SELECTED_USER
Value="USER_ID=user_911#FIRST_NAME=John#LAST_NAME=Smith"/>
<SELECTED_USER
Value="USER_ID=user_101#FIRST_NAME=John#LAST_NAME=Doe"/>
</PARAMETERS>
```

## Required Attributes

The following are the required attributes for a Table.

Table 56: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |

## Optional Attributes

The following are the optional attributes for a Table.

Table 57: Optional Attributes

| Attribute | Description |
|---|---|
| collapsable | Set to true if container can be collapsable. Default value is false. |
| containerId | Identifier for the container. |
| displayTextNoRecords | Message to be displayed when no records found for ui:table. Default text is "No records found" |
| document | Not used. Note: Attribute is deprecated. |
| doPaging | Turns paging on for the table. This is only relevant when maxRows is set to a value less than totalRowCount. |
| freeze | Specify number of columns to be frozen. If freeze is set to 2, first two columns will be frozen. |
| help | URL link to location of help page. |
| invert | Set to true if table should be inverted. The fields will represent rows and the data will grow the number of columns. Default value is false. |
| linked | Not used. Default value =”true” |
| maxRows | Set number of rows to be displayed at a given time. This is relevant only if paging is turned on. |
| method | Specify the parameter-passing style. Possible values include 'GET' and 'POST'. Default value is 'POST'. |
| postRowIdAs | This field is posted in the output variable to represent the row selected by the user |
| rememberSelections | If set to “true”, it will retain the selections in a paginated table across pages. |
| rowId | Logical Name of column of document to identify record in post processing. Primary keys are preferred candidate for this attribute. |
| rowSrc | An XPath expression used for culling the data for the table. The value is automatically evaluated as an expression |
| rowSubTreeSrc | An XPath expression used for culling the data for the treecell component in table. The value is automatically evaluated as an expression. |
| rowTreeSrc | An XPath expression used for culling the data for the treecell component in table. The value is automatically evaluated as an expression. |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| select | An XPath expression used for culling the data for the table. The value is automatically evaluated as an expression. |
| selectable | This field if set allows to select one or more rows of a table. If set to "Single" allows selection of only one row of a table by radio button, while if set to "All" allows multiple row selection of a table with checkbox. |

Table 57: Optional Attributes

| Attribute | Description |
|---|---|
| sortBy | Specify the field or column to sort table data on |
| sortOrder | Specify the sort order (Ascending or Descending) |
| startAtRow | Starting row number from which records should be retrieved. This field can be used with GET_DOCUMENT to achieve pagination. |
| title | Text to be displayed as the title of the container. Automatically internationalized. |
| totalRowCount | Specify the total number of rows shown in the table. This is required if paging is turned on. |
| urlParam | Not used. Default value=“true”. |
| draggable | Used to resize the width of the column. For example: If ui:table has freeze =“2” draggable =“true” then third column will have a draggable symbol using which you can resize the width of the column. |
| freezewidth | Set width for freeze table. |

### Used By

ui:cell, ui:page-group, ui:step

### Content

ui:buttons, ui:field-group, ui:pagination, ui:script, ui:validation

# Treecell

The Treecell is used with the table.

### Syntax

The following is the syntax for a treecell:

```
<ui:treecell>
```

### Example

The following are the example of using a treecell within a table.

```
<ui:table displayText="Orders" displayTextNoRecords="No Orders"
select="CUSTOMER_ORDERS" rowSubTreeSrc="ORDER_LINE_ITEMS/
ORDER_LINE_ITEM" collapsable="false">
<ui:field-group>
<ui:treecell displayText="Order ID" name="ID"
noChildren="{count(ORDER_LINE_ITEMS/ORDER_LINE_ITEM) = 0}"
depth="{count(ancestor::node()[name()='ORDER_LINE_ITEMS'])}" column="0">
<ui:url page="{$pages:standard_order_select_order_view}">
```

```
  <ID select="ID/@Value"/>
</ui:url>
</ui:treecell>
<ui:display-field displayText="Customer Name" name="CUST_ORG_NAME"/>
...
</ui:field-group>
</ui:table>
```

## Example Data

Refer to the *Data from Example Project* section.

## Required Attributes

The following are the required attributes for Tree Cell.

Table 58: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form |

## Optional Attributes

The following are the optional attributes for a Tree Cell.

Table 59: Optional Attributes

| Attribute | Description |
|---|---|
| column | Minimum number of columns the fields should be divided between. |
| depth | Sets the number of level in a tree cell. |
| noChildren | Sets the number of children for the tree node. Default value is true. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |

Table 59: Optional Attributes

| Attribute | Description |
|---|---|
| displayPage | Sets the URL for a link. Note: Attribute is deprecated. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Set to “true” if header should be editable. Default value is false. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |

Table 59: Optional Attributes

| Attribute | Description |
|---|---|
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

## Used By

ui:field-group, ui:otherwise, ui:when

## Content

ui:compare-validator, ui:links, ui:url

# IFrame

The <iframe> tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document.

## Syntax

The following is the syntax for iframe:

```
<ui:iframe>
```

## Example

The following are the example of using an iframe tag within a container.

```
<ui:container id="Outer" showFavorite="true" scrollable="true">
   <ui:step id="OuterStep" displayText="IFrame Example">
     <ui:iframe id="teradata" src="http://www.teradata.com"/>
   </ui:step>
 </ui:container>
```

## Example Data

None

## Required Attributes

The following are the required attributes for IFrame.

Table 60: Required Attributes

| Attribute | Description |
|---|---|
| id | Specify the Id of an iframe |
| name | Specify the name of the iframe |

## Optional Attributes

The following are the optional attributes for IFrame.

Table 61: Optional Attributes

| Attribute | Description |
|---|---|
| sandbox | Enables a set of extra restrictions for the content in the iframe. |
| seamless | Specifies that the iframe should look like it is a part of the containing document. |
| src | Specifies the address of the document embeded in the iframe. |
| srcdoc | Specifies the HTML content of the page to show within the iframe. |

## Used By

ui:step

## Content

None

# CHAPTER 5 Advanced Controls

### What's In This Chapter

This chapter gives you information about X-Document.

Topics include:


- Introduction
- Grid
- Row
- Cell
- Script
- Choose Field
- When
- Otherwise
- Dynamic PGL
- Excel Pivot
- Horizontal Tree
- Legend Field
- Pivot
- Pivot Data
- Pivot Field
- Spreadsheet
- Spreadsheet Field
- Spreadsheet Row
- Tree Node
- Vertical Tree
- Workbook
- Shell
- DIV


# Introduction

Every component in this chapter is divided into the following sections:

- **Description**. It gives you a brief description of the component.

- **Syntax**. It gives you the syntax of the component.
- **Example**. It gives you an example of how the component can be used.
- **Example Data**. It gives the example data for the component.
  - **Data from Existing Example**. It gives the data from the existing example for the component.
  - **Data from Example Project**. It gives the data from the example project available with the MDM Studio.
- **Required Attributes**. These are the list of required attributes for the component.
- **Optional Attributes**. These are the list of optional attributes for the component.
- **Content**. It gives the list of components that a component can contain.
- **Used By**. It gives the list of other components that can use this component.

# Grid

A grid is a layout manager that lays out a container's components in a rectangular grid. The container is divided into rectangles, and one component is placed in each rectangle. Some components may span multiple rectangle areas.

## Syntax

The following is the syntax for Grid.

```
<ui:grid>
```

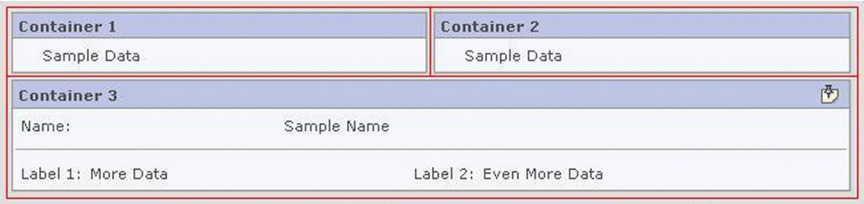
## Example

The following is an example of using a grid, to layout three containers on a page.

```
<ui:grid>
<ui:row>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
<ui:row>
<ui:cell colSpan="2">
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
</ui:grid>
```

The following figure illustrates the preceding code.

Figure 62: Grid




## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Grid.

Table 62: Optional Attributes

| Attribute | Description |
|---|---|
| cellSpacing | Default is ‘0’. |

## Used By

ui:cell, ui:page-group, ui:step

## Content

ui:row

# Row

It defines a row of cells with a grid.

## Syntax

The following is the syntax for Row.

```
<ui:row>
```

## Example

The following is an example of using two rows to layout three containers on a page.

```
<ui:grid>
<ui:row>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
<ui:row>
<ui:cell colSpan="2">
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
</ui:grid>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

None.

### Used By

ui:grid

### Content

ui:button, ui:cell, ui:vr

## Cell

It is used in conjuction with a grid to aid in page layout. Similar to a TD tag in HTML.

### Syntax

The following is the syntax for Cell.

```
<ui:cell>
```

## Example

The following is an example of using three cells to layout three containers on a page.

```
<ui:grid>
<ui:row>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
<ui:cell>
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
<ui:row>
<ui:cell colSpan="2">
<ui:container editable="false">
  ...
</ui:container>
</ui:cell>
</ui:row>
</ui:grid>
```

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Cell.

Table 63: Optional Attributes

| Attribute | Description |
|---|---|
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the contents should straddle. Default is 1. |
| rowSpan | Determines how many rows the contents should straddle. Default is 1. |
| valign | Vertically align this cell's contents to the top, center, bottom, or baseline of the cell. |
| width | Sets the width of this cell to n pixels or a percentage of the grid width. |

## Used By

ui:row

## Content

ui:buttons, ui:container, ui:field-group, ui:grid, ui:search, ui:table, ui:vr

# Script

It defines a script that should be included in the generated page.

## Syntax

The following is the syntax for Script.

```
<ui:script>
```

## Example

**Example1**: Usage of Third-party or Custom JavaScript on PGL UI

Custom JavaScript functions can be invoked using 'src' attribute of <ui:script> tag in PGL. The path of the JavaScript file (containing the JavaScript functions) should be given as below:

```
<ui:script id="script3" src="/td/javascript/test.js" />
```

The JavaScript needs to be included under mdmclient, that is, the MDM Web component.

Similarly, you may provide complete path of the context root url for the JavaScript file.

that is, `<ui:script id="script3" src="http://localhost:8080/mdm/td/javascript/test.js" />`

**Example 2**: The following is an example where a javascript is invoked when user changes the value in a dropdown.

The onChange attribute is set to invoke a javascript function called dropdownChanged.

```
<ui:dropdown name="DROP_DOWN" displayText="Dropdown" selectOne="true"
optionSrc="$root/OPTIONS/OPTION"
onChange="javascript:dropdownChanged();"/>
<ui:button name="DROPDOWN_CHANGED" displayText=""
displayCondition="false"/>
The javascript function is defined as shown below.
<ui:script>
function dropdownChanged(){
document.form.BUTTON_ID.value = 'DROPDOWN_CHANGED';
document.form.action = 'view.x2ps';
document.form.submit();
}
</ui:script>
```

When user changes dropdown value, the javascript is invoked and the BUTTON_ID value is set to DROPDOWN_CHANGED (a hidden button). This is equivalent to user selecting a button called DROPDOWN_CHANGED manually after changing dropdown value.

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Script.

Table 64: Optional Attributes

| Attribute | Description |
|---|---|
| Select | An XPath expression used for adding script from data. The field is automatically evaluated as an expression. If the select attribute is used, the body contents will be ignored. |
| Src | URL link for the javascript file. |
| type | Specify the type of field to display. Possible types include 'Display'. |
| Id | Identifier for HTML component. |
| displaytext | Text to be displayed as the label or header of the field. Automatically internationalized. |
| displaycondition | Conditional that determines if the field or column should be displayed or not. |
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |

## Used By

ui:button, ui:page-group, ui:step, ui:table

## Content

Any Component.

# Choose Field

It is a field component that allows a heterogeneous mix of other components in a table column.

## Syntax

The following is the syntax for Choose Field.

```
<ui:choose-field>
```

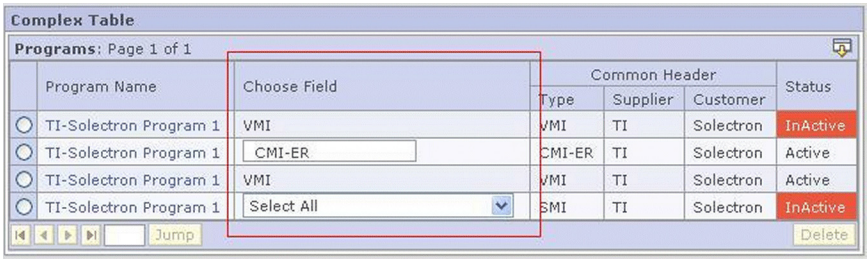
## Example

The following is an example of a choose field within a complex table. The choose-field shows heterogeneous fields within a single column.

```
<ui:table displayText="Programs" scrollable="false"
displayTextNoRecords="No Programs found" select="TPAS" rowSrc="TPA"
collapsable="false" selectable="Single">
<ui:help url="{$pages:search_results_help}"/>
<ui:field-group>
<ui:link-field name="TPA_NAME" displayText="Program Name">
<ui:url page="{$pages:cpm_program_review}">
  <TPA_ID select="TPA_ID/@Value"/>
</ui:url>
</ui:link-field>
<ui:choose-field name="EVENT_ID" displayText="Choose Field">
<ui:when test="TPA_TYPE/@Value = 'VMI'">
<ui:display-field displayText="" name="TPA_TYPE"/>
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'CMI-ER'">
<ui:entry-field displayText="" name="TPA_TYPE" value="{TPA_TYPE/
@Value}"/>
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'SMI'">
<ui:dropdown displayText="Current Forecast State" name="FCST_STATE"
selectAll="true" dataType="Constant">
  <ui:option id="VALIDATION_FAILED"
  value="VALIDATION_FAILED"/>
  <ui:option id="MODIFIED" value="MODIFIED"/>
  <ui:option id="RELEASE_EXTRACTION_FAILED"
  value="RELEASE_EXTRACTION_FAILED"/>
</ui:dropdown>
</ui:when>
<ui:otherwise>
<ui:display-field displayText="" name="TPA_TYPE"/>
</ui:otherwise>
</ui:choose-field>
...
</ui:field-group>
</ui:table>
<ui:field-group columns="10">
<ui:legend-field name="ACCEPT" id="ACCEPT" displayText="Accept"
color="#66CC66" />
<ui:legend-field name="REJECT" id="REJECT" displayText="Reject"
color="#FF0000" />
</ui:field-group>
```

The following figure shows a choose-field column within a complex table.

Figure 63: Choose Field




## Example Data

### Data from Example Project

**Workflow**: AdvancedControls

**Step**: Conditional

**PGL:**

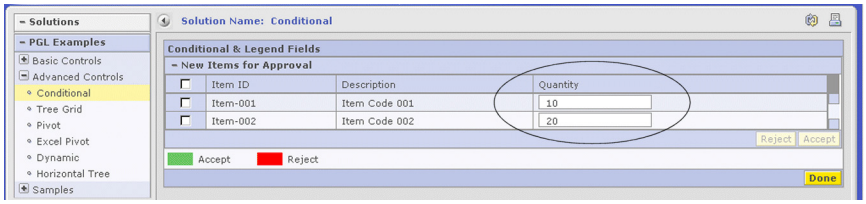
```
<ui:table id="itemsTable" displayText="New Items for Approval"
select="$root/ITEMS_LIST" rowId="ITEM_ID" rowSrc="ITEM" selectable="All"
postRowIdAs="SELECTED_ITEM" maxRows="-1">
<ui:field-group id="myFieldGroup">
<ui:display-field name="ITEM_ID" displayText="Item ID" style="{ITEM_ID/
@Style}"/>
<ui:display-field name="DESC" displayText="Description"/>
<ui:choose-field name="chooseField" displayText="Quantity">
<ui:when test="QTY/@Type='Entry'">
  <ui:entry-field name="QTY"/>
</ui:when>
<ui:otherwise>
  <ui:display-field name="QTY"/>
</ui:otherwise>
</ui:choose-field>
</ui:field-group>
<ui:buttons>
<ui:button displayText="Reject" id="REJECT" enabledBySelections="true"/>
<ui:button displayText="Accept" id="ACCEPT" enabledBySelections="true"/>
<ui:button displayText="Reset" id="RESET"
displayCondition="{count($root/ITEMS_LIST/ITEM[QTY/@Type='Display'])>0}"
enabledBySelections="true"/>
</ui:buttons>
</ui:table>
```

PreProcessing:

```
<TO_DOCVAR AssignToVar="inXml">
<ROOT>
<ITEMS_LIST TotalRowCount="2">
<ITEM>
  <ITEM_ID Value="Item-001"/>
  <DESC Value="Item Code 001"/>
  <QTY Value="10" Type="Entry"/>
</ITEM>
<ITEM>
  <ITEM_ID Value="Item-002"/>
```

```
  <DESC Value="Item Code 002"/>
  <QTY Value="20" Type="Entry"/>
</ITEM>
</ITEMS_LIST>
</ROOT>
</TO_DOCVAR>
```

Figure 64: Choose Field Example




## Required Attributes

The following are the required attributes for Choose Field.

Table 65: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. |

## Optional Attributes

The following are the optional attributes for Choose Field.

Table 66: Optional Attributes

| Attribute | Description |
|---|---|
| editable | Set to true if field should be editable. Default value is true. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |

Table 66: Optional Attributes

| Attribute | Description |
|---|---|
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | URL link to the location of help page. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Possible value is T_FIELD_ENTRY. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Specify a style controlling the appearance of header cell's contents. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |

Table 66: Optional Attributes

| Attribute | Description |
|---|---|
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:field-group

### Content

ui:otherwise, ui:when, ui:compare-validator, ui:links, ui:url

# When

It is used in conjuction with the ui:choose-field. It is equivalent to the Java case statement.

### Syntax

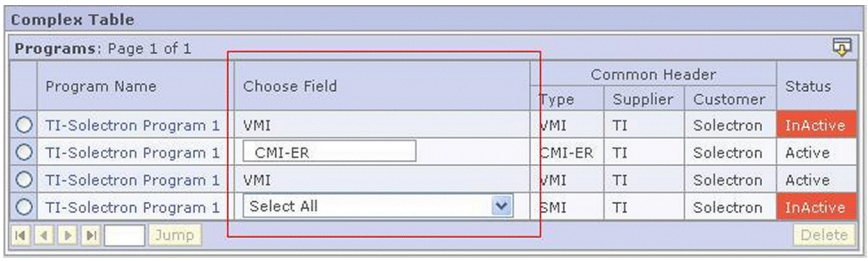
The following is an example of a choose field within a complex table. The choose-field shows heterogeneous fields within a single column.

```
<ui:table displayText="Programs" scrollable="false"
displayTextNoRecords="No Programs found" select="TPAS" rowSrc="TPA"
collapsable="false" selectable="Single">
<ui:help url="{$pages:search_results_help}"/>
<ui:field-group>
...
<ui:choose-field name="EVENT_ID" displayText="Choose Field">
<ui:when test="TPA_TYPE/@Value = 'VMI'">
   <ui:display-field displayText="" name="TPA_TYPE"/>
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'CMI-ER'">
   <ui:entry-field displayText="" name="TPA_TYPE"
   value="{TPA_TYPE/@Value}"/>
</ui:when>
<ui:when test="TPA_TYPE/@Value = 'SMI'">
   <ui:dropdown displayText="Current Forecast State"
   name="FCST_STATE" selectAll="true"
   dataType="Constant">
      <ui:option id="VALIDATION_FAILED"
      value="VALIDATION_FAILED"/>
      <ui:option id="MODIFIED" value="MODIFIED"/>
      <ui:option id="RELEASE_EXTRACTION_FAILED"
      value="RELEASE_EXTRACTION_FAILED"/>
   </ui:dropdown>
</ui:when>
<ui:otherwise>
   <ui:display-field displayText="" name="TPA_TYPE"/>
```

```
</ui:otherwise>
</ui:choose-field>
...
</ui:field-group>
</ui:table>
```

The following figure shows a choose-field column within a complex table.

Figure 65: Where




## Example

Refer to the example in the *Choose Field* section.

## Example Data

The following are the required attributes for Choose Field.

## Required Attributes

The following are the required attributes for Choose Field.

Table 67: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form. |

## Optional Attributes

The following are the optional attributes for Choose Field.

Table 68: Optional Attributes

| Attribute | Description |
|---|---|
| test | An XPath expression that evaluates as a conditional. If the expression evaluates to true, then the child fields are included. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |

Table 68: Optional Attributes

| Attribute | Description |
|---|---|
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date', ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | URL link to the location of the help page. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Possible value is T_FIELD_ENTRY. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Specify a style controlling the appearance of header cell's contents. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |

Table 68: Optional Attributes

| Attribute | Description |
|---|---|
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:choose-field

### Content

ui:button, ui:checkbox, ui:date-range, ui:display-field, ui:dropdown, ui:entry-field, ui:field-group, ui:field-space, ui:hidden-field, ui:image-link-field, ui:link-field, ui:radiobutton, ui:textarea, ui:treecell, ui:compare-validator, ui:links, ui:url

# Otherwise

It is used in conjuction with the ui:choose-field. It is equivalent to the Java default statement in a switch.

### Syntax

The following is the syntax for Otherwise.

```
<ui:otherwise>
```

### Example Data

Refer to the example in the *Choose Field* section.

### Required Attributes

The following are the required attributes for Otherwise.

Table 69: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form |

## Optional Attributes

The following are the optional attributes for Otherwise.

Table 70: Optional Attributes

| Attribute | Description |
|---|---|
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |
| displayPage | URL link to the location of the help page. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerEditable | Possible value is T_FIELD_ENTRY. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Specify a style controlling the appearance of header cell's contents. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |

Table 70: Optional Attributes

| Attribute | Description |
|---|---|
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

## Used By

ui:choose-field

## Content

ui:button, ui:checkbox, ui:date-range, ui:display-field, ui:dropdown, ui:entry-field, ui:field-group, ui:field-space, ui:hidden-field, ui:image-link-field, ui:link-field, ui:radiobutton, ui:textarea, ui:treecell, ui:compare-validator, ui:links, ui:url

# Dynamic PGL

It allows PGL content to be passed in via the source data. The developer can use x2 commands to dynamically construct the PGL data. Most inefficient and should be used sparingly

## Syntax

The following is the syntax for Dynamic PGL.

```
<ui:dynamic-pgl>
```

## Example

```
<ui:container id="topContainer">
      <ui:step id="step" displayText="Dynamic PGL">
         <ui:dynamic-pgl select="$root/PGL/*"/>
         <ui:buttons>
            <ui:button id="TABLE" name="TABLE" displayText="Table View"
            disabled="{$root/DATA/VIEW_TYPE/@Value='Table'}"/>
            <ui:button id="CHART" name="CHART" displayText="Chart View"
            disabled="{$root/DATA/VIEW_TYPE/@Value='Chart'}"/>
            <ui:divider/>
            <ui:button id="DONE" name="DONE" displayText="Done"
            emphasized="true"/>
         </ui:buttons>
      </ui:step>
</ui:container>
```

## Example Data

### Data from Example Project

**Workflow**: AdvancedControls

Step: Dynamic

PreProcessing:

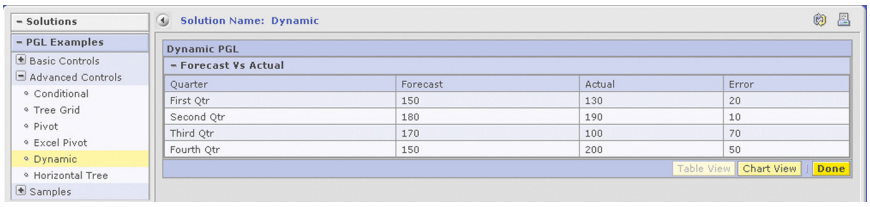
```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <PGL/>
      <DATA/>
    </SAMPLE_XML>
</TO_DOCVAR>
<IF_TEST Test="$pglType='Table'">
    <THEN>
      <APPEND_TO_XML SelectList="$inXml/PGL">
       <ui:table xmlns:ui="http://www.teradata.com/ui" id="dynamicTable"
displayText="Forecast Vs Actual" rowSrc="$root/DATA/LINE/ROW" maxRows="-
1" scrollable="false">
          <ui:field-group id="tableColumns">
            <ui:display-field name="QUARTER" displayText="Quarter"/>
            <ui:display-field name="FORECAST" displayText="Forecast"/>
            <ui:display-field name="ACTUAL" displayText="Actual"/>
            <ui:display-field name="ERROR" displayText="Error"/>
```

```
          </ui:field-group>
        </ui:table>
      </APPEND_TO_XML>
    </THEN>
    <ELSE>
      <APPEND_TO_XML SelectList="$inXml/PGL">
        <ui:chart xmlns:ui="http://www.teradata.com/ui" width="500"
height="170" xLabelsWidth="100" yAxisNumTicks="4" displayText="Forecast
Vs Actual by Quarter" select="$root/DATA/LINE">
          <ui:chart-field displayText="Quarter" name="QUARTER"/>
          <ui:chart-field displayText="Forecast" name="FORECAST"/>
          <ui:chart-field displayText="Actual" name="ACTUAL"/>
          <ui:chart-field displayText="Forecast Error" name="ERROR"/>
        </ui:chart>
      </APPEND_TO_XML>
    </ELSE>
</IF_TEST>
<APPEND_TO_XML Select="$inXml/DATA">
    <LINE>
      <ROW>
        <QUARTER Value="First Qtr"/>
        <FORECAST Value="150"/>
        <ACTUAL Value="130"/>
        <ERROR Value="20"/>
      </ROW>
      <ROW>
        <QUARTER Value="Second Qtr"/>
        <FORECAST Value="180"/>
        <ACTUAL Value="190"/>
        <ERROR Value="10"/>
      </ROW>
      <ROW>
        <QUARTER Value="Third Qtr"/>
        <FORECAST Value="170"/>
        <ACTUAL Value="100"/>
        <ERROR Value="70"/>
      </ROW>
      <ROW>
        <QUARTER Value="Fourth Qtr"/>
        <FORECAST Value="150"/>
        <ACTUAL Value="200"/>
        <ERROR Value="50"/>
      </ROW>
    </LINE>
    <VIEW_TYPE Value="{$pglType}"/>
</APPEND_TO_XML>
```

PostProcessing:

```
<IF_TEST Test="$userAction='CHART'">
    <THEN>
      <SET Var="pglType" FromValue="Chart"/>
    </THEN>
    <ELSE>
      <SET Var="pglType" FromValue="Table"/>
    </ELSE>
</IF_TEST>
```

Figure 66: Dynamic PGL Example




## Required Attributes

The following are the required attributes for Dynamic PGL.

Table 71: Required Attributes

| Attribute | Description |
|---|---|
| select | The following are the required attributes for Dynamic PGL. |

## Optional Attributes

The following are the optional attributes for Dynamic PGL.

## Used By

ui:step

## Content

None.

# Excel Pivot

Excel Pivot is the PGL plug-in for the Microsoft Excel PivotTable web component. The PivotTable allows the user to organize multi-dimensional data into many different views for increased analysis ability. Using the Excel plug-in, the user can change the PivotTable dimensions without having to make another request on the application server to recreate the data.

**Note:** Perform the following steps if you are unable to see the excel pivot tables in your web browser:

1) Enable ADODB.Stream from IE by going to regedit and removing the following entry

   `HKEY_LOCAL_MACHINE\SOFTWARE > Microsoft > Internet Explorer > ActiveX Compatibility > {00000566-0000-0010-8000-00AA006D2EA4}`

   Microsoft has removed ADODB.Stream from IE because of security reasons. Adodb.stream provides a method for reading and writing files on a hard drive. This by-design functionality is sometimes used by web applications. However, when combined with known security vulnerabilities in Microsoft Internet Explorer, it could allow an

internet web site to execute script from the Local Machine Zone (LMZ). This occurs because the ADODB.Stream object allows access to the hard drive when hosted within Internet Explorer. For more details please follow the link

http://support.microsoft.com/?kbid=870669

Clear you cache history from you Browser and open the browser again.

2) Ensure you have the Microsoft Web Components 2003 installed on your machine.

## Syntax

The following is the syntax for Excel Pivot.

```
<ui:excel-pivot>
```

## Example

```
<ui:excel-pivot id="pivot" displayText="Data" select="$root/
MULTI_COLLAB_DATA/ITEMS" rowSrc="ITEM" subRowSrc="SPLITPERCENTAGES/
SPLITPERCENTAGE">
      <ui:pivot-field axis="row" name="ITEMID" displayText="Item ID"/>
      <ui:pivot-field axis="row" name="MAXSPEED" displayText="Max
      Speed"/>
      <ui:pivot-field axis="row" name="DATEBUCKET" displayText="Date"
      dataType="Date"/>
      <ui:pivot-field axis="column" name="DLCPCATEGORY"
      displayText="DLCP Category"/>
      <ui:display-field name="INCPERCENTAGE" displayText="Inc
      Percentage" id="INCPERCENTAGE"/>
</ui:excel-pivot>
```

## Example Data

### Data from Example Project

**Workflow**: AdvancedControls

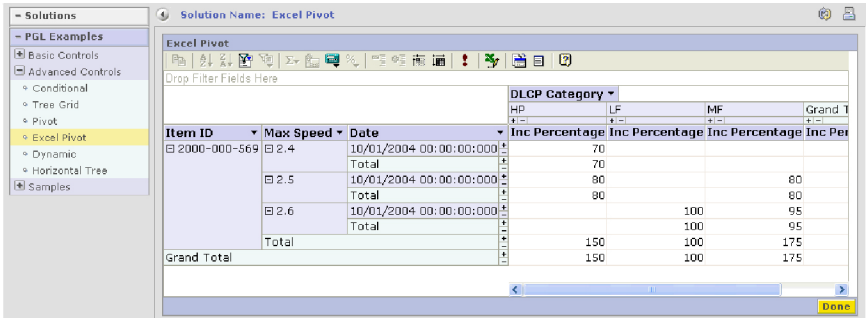
**Step**: ExcelPivot

**PreProcessing:**

```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <MULTI_COLLAB_DATA>
        <ITEMS>
          <ITEM>
            <ITEMID Value="2000-000-569"/>
            <MAXSPEED Value="2.6"/>
            <DATEBUCKET Value="10/01/2004 00:00:00:000"/>
            <SPLITPERCENTAGES>
              <SPLITPERCENTAGE>
                <DLCPCATEGORY Value="LF"/>
                <INCPERCENTAGE Value="100"/>
              </SPLITPERCENTAGE>
            </SPLITPERCENTAGES>
          </ITEM>
          <ITEM>
            <ITEMID Value="2000-000-569"/>
```

```
            <MAXSPEED Value="2.5"/>
            <DATEBUCKET Value="10/01/2004 00:00:00:000"/>
            <SPLITPERCENTAGES>
              <SPLITPERCENTAGE>
                <DLCPCATEGORY Value="HP"/>
                <INCPERCENTAGE Value="80"/>
              </SPLITPERCENTAGE>
            </SPLITPERCENTAGES>
          </ITEM>
          <ITEM>
            <ITEMID Value="2000-000-569"/>
            <MAXSPEED Value="2.4"/>
            <DATEBUCKET Value="10/01/2004 00:00:00:000"/>
            <SPLITPERCENTAGES>
              <SPLITPERCENTAGE>
                <DLCPCATEGORY Value="HP"/>
                <INCPERCENTAGE Value="70"/>
              </SPLITPERCENTAGE>
            </SPLITPERCENTAGES>
          </ITEM>
          <ITEM>
            <ITEMID Value="2000-000-569"/>
            <MAXSPEED Value="2.6"/>
            <DATEBUCKET Value="10/01/2004 00:00:00:000"/>
            <SPLITPERCENTAGES>
              <SPLITPERCENTAGE>
                <DLCPCATEGORY Value="MF"/>
                <INCPERCENTAGE Value="95"/>
              </SPLITPERCENTAGE>
            </SPLITPERCENTAGES>
          </ITEM>
          <ITEM>
            <ITEMID Value="2000-000-569"/>
            <MAXSPEED Value="2.5"/>
            <DATEBUCKET Value="10/01/2004 00:00:00:000"/>
            <SPLITPERCENTAGES>
              <SPLITPERCENTAGE>
                <DLCPCATEGORY Value="MF"/>
                <INCPERCENTAGE Value="80"/>
              </SPLITPERCENTAGE>
            </SPLITPERCENTAGES>
          </ITEM>
        </ITEMS>
      </MULTI_COLLAB_DATA>
    </SAMPLE_XML>
</TO_DOCVAR>
```

Figure 67: Excel Pivot




## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Excel Pivot.

Table 72: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| id | Identifier for HTML component. |
| rowSrc | An XPath expression used for selecting data for pivot data. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| subRowSrc | An XPath expression used for selecting data for pivot row data. |

## Used By

ui:step

## Content

ui:display-field, ui:pivot-field

# Horizontal Tree

Horizontal tree is a tree style browser that can be expanded horizontally. Each node in the tree can be a expanded to branch off into a hierarchical structure.

## Syntax

The following is the syntax for Horizontal tree.

```
<ui:horozontal-tree>
```

## Example

```
<ui:step id="horizontalTree" displayText="Horizontal Tree">
<ui:horizontal-tree id="myTree" displayText="Example" select="$root/
MY_TREE" selectedStyle="{background-color:yellow;}" width="100%"
collapsable="true" scrollable="true" imageInactive="true"/>
<ui:buttons>
<ui:button id="DONE" name="DONE" displayText="Done" emphasized="true"/>
</ui:buttons>
</ui:step>
```

## Example Data

### Data from Example Project

**Workflow**: AdvancedControls

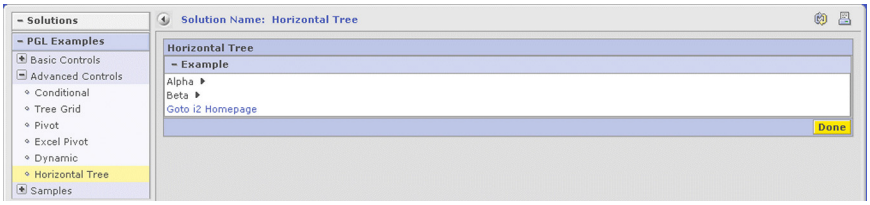
Step: Horizontal Tree

PreProcessing:

```
<IF_TEST Test="$thisParam/Column=null">
    <THEN>
      <TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <MY_TREE>
            <Column>
              <Row Id="1" Value="Alpha" Selected="false"/>
              <Row Id="2" Value="Beta" Selected="false"/>
              <Row Id="3" Value="Goto teradata Homepage" Leaf="true"
url="http://www.teradata.com"/>
            </Column>
          </MY_TREE>
        </SAMPLE_XML>
      </TO_DOCVAR>
    </THEN>
</IF_TEST>
<IF_TEST Test="$thisParam/Column/@Value=1 and $thisParam/Id/@Value=1">
    <THEN>
      <TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <MY_TREE>
            <Column>
              <Row Id="1" Value="Alpha" Selected="true"/>
              <Row Id="2" Value="Beta" Selected="false"/>
              <Row Id="3" Value="Goto teradata Homepage" Leaf="true"
url="http://www.teradata.com"/>
            </Column>
            <Column>
              <Row Id="4" Value="A-1" Leaf="true" url="/base/core/
home.jsp"/>
              <Row Id="5" Value="A-2" Selected="false"/>
            </Column>
```

```
          </MY_TREE>
        </SAMPLE_XML>
      </TO_DOCVAR>
    </THEN>
</IF_TEST>
<IF_TEST Test="$thisParam/Column/@Value=1 and $thisParam/Id/@Value=2">
    <THEN>
      <TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <MY_TREE>
            <Column>
              <Row Id="1" Value="Alpha" Selected="false"/>
              <Row Id="2" Value="Beta" Selected="true"/>
              <Row Id="3" Value="Goto teradata Homepage" Leaf="true"
url="http://www.teradata.com"/>
            </Column>
            <Column>
              <Row Id="6" Value="B-001" Leaf="true" url="/base/core/
home.jsp"/>
              <Row Id="7" Value="B-002" Leaf="true" url="/base/core/
home.jsp"/>
              <Row Id="8" Value="B-003" Leaf="true" url="/base/core/
home.jsp"/>
            </Column>
          </MY_TREE>
        </SAMPLE_XML>
      </TO_DOCVAR>
    </THEN>
</IF_TEST>
<IF_TEST Test="$thisParam/Column/@Value=2 and $thisParam/Id/@Value=5">
    <THEN>
      <TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <MY_TREE>
            <Column>
              <Row Id="1" Value="Alpha" Selected="true"/>
              <Row Id="2" Value="Beta" Selected="false"/>
              <Row Id="3" Value="Goto teradata Homepage" Leaf="true"
url="http://www.teradata.com"/>
            </Column>
            <Column>
              <Row Id="4" Value="A-1" Leaf="true" url="/base/core/
home.jsp"/>
              <Row Id="5" Value="A-2" Selected="true"/>
            </Column>
            <Column>
              <Row Id="9" Value="Show Basic Controls" Leaf="true" url="/
base/start.x2ps?START_WORKFLOW=basicControls&amp;Display=Basic"/>
            </Column>
          </MY_TREE>
        </SAMPLE_XML>
      </TO_DOCVAR>
    </THEN>
</IF_TEST>
```

Figure 68: Horizontal Tree Example




## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Horizontal Tree.

Table 73: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| collapsable | Set to true if container can be collapsable. Default value is false. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title. Automatically internationalized. |
| height | Sets the preferred height of the tree |
| id | Identifier for HTML Component. |
| imageActive | Sets the image icon for active tree nodes. Default value=”td/image/next_active.gif”. |
| imageInactive | Sets the image icon for inactive tree nodes. Default value=”td/image/next_inactive.gif”. |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| rowSrc | An XPath expression used for selecting data for pivot data. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| selectedStyle | Sets the style of the selected tree nodes. Default value=”{background-color:#E6E6E6;”. |
| style | Specify a style controlling the appearance of cell's contents. |
| width | Sets the preferred width of the tree. |

## Used By

ui:step

## Content

None.

# Legend Field

Legend field component is used to specify the legend to highlight any data component with the required background color.

## Syntax

The following is the syntax for Legend Field.

```
<ui:legend-field>
```

## Example Data

Refer to the example in the *Choose Field* section.

### Data from Example Project

**Workflow**: AdvancedControls

**Step**: Conditional

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes for Legend Field:

Table 74: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| color | Used to specify the color of the legend field. For example, color="#66CC66" |

## Used By

ui:page-group, ui:step

## Content

ui:field-group, ui:table

# Pivot

Pivot is the PGL version of the static HTML representation of a pivot table. You can determine the grouping of the dimension at design time and the data is presented in a static form to the user.

## Syntax

The following is the syntax for Pivot.

```
<ui:pivot>
```

## Example

```
<ui:pivot id="pivot" displayText="Item Demands by Region" select="$root/
DATA" rowSrc="ITEM" subRowSrc="DEMANDS/DEMAND" selectable="Single"
maxRows="-1">
<ui:pivot-field axis="row" name="ID" displayText="Item ID" style="{ID/
@Style}"/>
<ui:pivot-field axis="row" name="Description" displayText="Description"/
>
<ui:pivot-field axis="column" name="BUCKET_DATE" colSpan="2"/>
<ui:pivot-field axis="column" name="COLUMN_LABEL"/>
<ui:display-field name="AMERICAS" displayText="Americas"
dataType="Number" style="{AMERICAS/@Style}"/>
<ui:display-field name="EMEA" displayText="Europe, Middle-east &amp;
Africa" dataType="Number" style="{EMEA/@Style}"/>
<ui:display-field name="GAP" displayText="Greater Asia Pacific"
dataType="Number" style="{GAP/@Style}"/>
</ui:pivot>
```

## Example Data

### Data from Example Project

**Workflow**: AdvancedControls

**Step**: Pivot

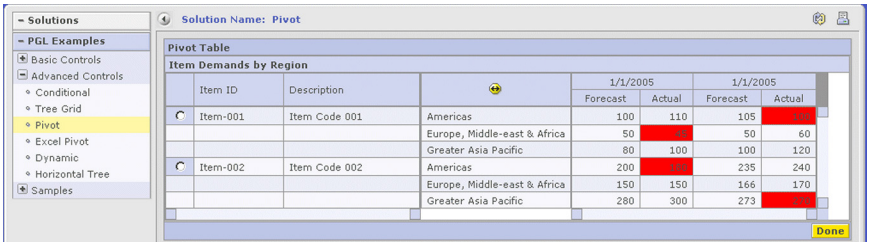
**PreProcessing:**

```
<TO_DOCVAR AssignToVar="inXml">
    <SAMPLE_XML>
      <DATA>
        <ITEM>
          <ID Value="Item-001"/>
```

```
  <Description Value="Item Code 001"/>
  <DEMANDS>
    <DEMAND>
      <AMERICAS Value="100"/>
      <EMEA Value="50"/>
      <GAP Value="80"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Forecast"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="110"/>
      <EMEA Value="45" Style="{{background:red}}"/>
      <GAP Value="100"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Actual"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="105"/>
      <EMEA Value="50"/>
      <GAP Value="100"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Forecast"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="100" Style="{{background:red}}"/>
      <EMEA Value="60"/>
      <GAP Value="120"/>
      <BUCKET_DATE Value="2/1/2005"/>
      <COLUMN_LABEL Value="Actual"/>
    </DEMAND>
  </DEMANDS>
</ITEM>
<ITEM>
  <ID Value="Item-002"/>
  <Description Value="Item Code 002"/>
  <DEMANDS>
    <DEMAND>
      <AMERICAS Value="200"/>
      <EMEA Value="150"/>
      <GAP Value="280"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Forecast"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="180" Style="{{background:red}}"/>
      <EMEA Value="150"/>
      <GAP Value="300"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Actual"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="235"/>
      <EMEA Value="166"/>
      <GAP Value="273"/>
      <BUCKET_DATE Value="1/1/2005"/>
      <COLUMN_LABEL Value="Forecast"/>
    </DEMAND>
    <DEMAND>
      <AMERICAS Value="240"/>
```

```
                    <EMEA Value="170"/>
                    <GAP Value="270" Style="{{background:red}}"/>
                    <BUCKET_DATE Value="2/1/2005"/>
                    <COLUMN_LABEL Value="Actual"/>
                  </DEMAND>
                </DEMANDS>
              </ITEM>
            </DATA>
          </SAMPLE_XML>
</TO_DOCVAR>
```

Figure 69: Pivot Example




## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 75: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| configurable | Set to true if data columns can be configured. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |
| exportable | Set to True for Export to Excel functionality. |
| id | Identifier for HTML Component. |
| maxRows | Set number of rows to be displayed at a given time. This is relevant only if paging is turned on. |
| rowSrc | An XPath expression used for selecting data for pivot data. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |

Table 75: Optional Attributes

| Attribute | Description |
|---|---|
| selectable | This field if set allows to select one or more rows of a table. If set to "Single" allows selection of only one row of a table by radio button, while if set to "All" allows multiple row selection of a table with checkbox. |
| selectColumnText | Sets the display text for the single selector column. |
| startAtRow | Starting row number from which records should be retrieved. This field can be used with GET_DOCUMENT to achieve pagination. |
| subRowSrc | An XPath expression used for selecting data for pivot row data. |
| totalRowCount | Sets the total number of rows for paged table. Used to show number of table pages. Default value=”@TotalRowCount”. |

### Used By

ui:page-group, ui:step

### Content

ui:choose-field, ui:display-field, ui:entry-field, ui:image-link-field, ui:link-field, ui:pivot-data, ui:pivot-field

# Pivot Data

PivotData is used in conjunction with the HTML Pivot component to specify the data measures to be displayed on the right-hand side of the pivot table.

### Syntax

The following is the syntax for Pivot Data.

```
<ui:pivot-data>
```

### Example Data

Data is not required for this component.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 76: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| axis | Set to “column” if field is pivoted. There must be one and only one pivot-field with a column axis. Set to “row” if column is to be displayed on left side of pivot table. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |

## Used By

ui:pivot

## Content

ui:display-field

# Pivot Field

Pivot Field is used in conjunction with the HTML Pivot component to specify the data to be pivoted. The PivotField components are listed in the order that they will be grouped from left to right in the pivot table.

## Syntax

The following is the syntax for Pivot Field.

```
<ui:pivot-field>
```

### Example

Refer to the example in the *Pivot* section.

## Example Data

Refer to the example in the *Pivot* section.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 77: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| axis | Set to “column” if field is pivoted. There must be one and only one pivot-field with a column axis. Set to “row” if column is to be displayed on left side of pivot table. |
| colSpan | Determines how many columns the contents should straddle. Default is 1. |
| dataType | Specify the type of data for internationalization. Possible types include 'Text', 'Number', 'Date' , ‘DateTime’ and 'Currency'. Default value is 'Text'. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |
| name | Specifies the name of the parameter that is submitted as part of the form. |
| onClick | Specify the javascript to execute if the field is clicked. |
| selectable | This field if set allows to select one or more rows of a table. If set to "Single" allows selection of only one row of a table by radio button, while if set to "All" allows multiple row selection of a table with checkbox. |
| style | Specify a style controlling the appearance of cell's contents. |

### Used By

ui:excel-pivot, ui:pivot

### Content

None.

# Spreadsheet

The spreadsheet is an extension of "table" and the workbook component is a container to host multiple "spreadsheets".

### Syntax

The following is the syntax for Spreadsheet.

```
<ui:spreadsheet>
```

### Example Data

Refer to the *Example Data* section.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 78: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| protect | Text to be displayed as the title of the pad. Automatically internationalized. |
| select | An XPath expression used for culling the data for the field groups children. The value is automatically evaluated as an expression. |
| sheetName | Default is 'Worksheet'. |

### Used By

ui:workbook

### Content

ui:spreadsheet-data-row, ui:spreadsheet-row

## Spreadsheet Field

The spreadsheet field is equivalent to a table field.

### Syntax

The following is the syntax for Spreadsheet Field.

```
<ui:spreadsheet-field>
```

### Example Data

Refer to the *Example Data* section.

### Required Attributes

The following are the required attributes:

Table 79: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |
| name | Specify the name of the parameter that is submitted as part of the form |

## Optional Attributes

The following are the optional attributes.

Table 80: Optional Attributes

| Attribute | Description |
|---|---|
| cellName | The cell name of the field. |
| dataType | Specifies the type of the data, possible types include: text, number, data DateTime and currency. Default value is text. |
| editable | Set to true if field should be editable. Default value is true. |
| formula | The formula to set on the field. |
| name | Specifies the name of the parameter that is submitted as part of the form. |
| repeat | Set to “true” is cell should be repeated. always set to “true”. |
| style | The cell style for the field. |
| value | Sets the initial value for the field. |
| disabled | Set to true if field should be disabled. Default value is false. |
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| align | Sets the horizontal alignment for field data. Possible values include 'left', 'center' and 'right'. Default value is data type specific according to Teradata standards. |
| class | Specify a style class controlling the appearance of cell's contents. |
| colSpan | Determines how many columns the field should straddle. Default is 1. |
| configurable | Set to “true” if field can be configurable. The table must be configurable for the field to be configurable. Default value is “true”. |
| decimals | Number of fraction digits to be displayed. |
| defaultValue | Default value used if no value is initially given. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |
| displayNoWrap | Set to true if no wrapping is desired within the field. |

Table 80: Optional Attributes

| Attribute | Description |
|---|---|
| displayPage | URL link value which might refer to pages.cnd definition or user defined http URL. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| elementName | Possible value is T_FIELD_ENTRY. |
| filter | Set to “true” if field can be filtered. The table must be filtered for the field to be filtered. Default value is “true”. |
| format | Format to use other than the default. |
| headerClass | Specify a style class controlling the appearance of header cell's contents. |
| headerDataType | Sets the data type for the header. Used for internationalization. |
| headerEditable | Set to “true” if header should be editable. Default value is “false”. |
| headerElementName | Possible value is T_FIELD_ENTRY. |
| headerLink | Sets the URL for the header. |
| headerStyle | Specify a style controlling the appearance of header cell's contents. |
| headerType | Specifies the type of header field to display. Possible types include 'Display'. Default value is 'Display'. |
| headerUom | Sets the unit of measure at the header level. |
| inFormat | Format of the date data sent to page, if not MM/dd/yyyy. |
| onClick | Specify the javascript to execute if the field is clicked. |
| required | Set to true if data must be entered into the field. The indication of a required field will be a red asterisk displayed next to the label. Default value is false. |
| rowSpan | Determines how many rows the field should straddle. Default is 1. |
| select | An XPath expression used for populating the field value. The field is automatically evaluated as an expression. Default is to derived XPath expression from element name's Value attribute. An example is NAME/@Value. |
| size | Sets the size of the field. |
| sortable | Set to true if table column can be sorted. Sort functionality and icons will be activated. |
| style | Specify a style controlling the appearance of cell's contents. |
| type | Specify the type of field to display. Possible types include 'Display'. Default value is 'Display'. |

Table 80: Optional Attributes

| Attribute | Description |
|---|---|
| unFormat | If it is date, then unformat = “true” means it will display the MDM default date format. If unformat = “false” means it will display the current language format. Most of the time, the unformatting is done automatically. Advanced use only.<br>If it is a string, then unformat = “true” means it will not display i18nized value of that property, if unformat = “false” means it will display i18nized value of that property. |
| uom | Sets the unit of measure for the field. |
| uploadType | Sets the upload type for the field. Set to “file” if the uploaded item is a file. |
| value | Initial value set to the field. |
| valuePrepend | String to be prepended to the value before an i18n lookup. |

### Used By

ui:spreadsheet-row

### Content

ui:compare-validator, ui:custom-validator, ui:links, ui:url

# Spreadsheet Row

The spreadsheet row is an extended component similar to a table row.

### Syntax

The following is the syntax for Spreadsheet Row.

```
<ui:spreadsheet-row>
```

### Example Data

Refer to the *Example Data* section.

### Required Attributes

None.

### Optional Attributes

The following are the optional attributes:

Table 81: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| rowSrc | An XPath expression used for selecting the data for the row. The value is automatically evaluated as an expression. |

## Used By

ui:spreadsheet

## Content

ui:spreadsheet-field

# Tree Node

Tree Node is used in conjunction with the vertical tree browser. Each node is defined as a link and returns an ID that can be used in the workflow.

## Syntax

The following is the syntax for Tree Node.

```
<ui:tree-node>
```

### Example

Refer to the example in the *Vertical Tree* section.

## Example Data

Refer to the example data in the *Vertical Tree* section.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 82: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| id | Default is 'ID'. |
| select | An XPath expression used for selecting the node data. The field is automatically evaluated as an expression. |
| url | Set the URL value for a link. |

### Used By

ui:tree-node, ui:vertical-tree

### Content

ui:tree-node

# Vertical Tree

Vertical tree is a tree style browser that can be expanded vertically (similar to a windows explorer). Each node can be expanded to branch off into a hierarchical structure.

### Syntax

The following is the syntax for Vertical Tree.

```
<ui:vertical-tree>
```

### Example

```
<ui:step displayText="Tree" id="Notes">
      <ui:vertical-tree id="myTree" title="A Tree" select="$root/
      RESPONSE" displayText="My Tree" postAction="true" width="200"
      height="200">
         <ui:tree-Node id="Specification" select="ROWS"/>
      </ui:vertical-tree>
</ui:step>
```

### Example Data

#### Data from Example Project

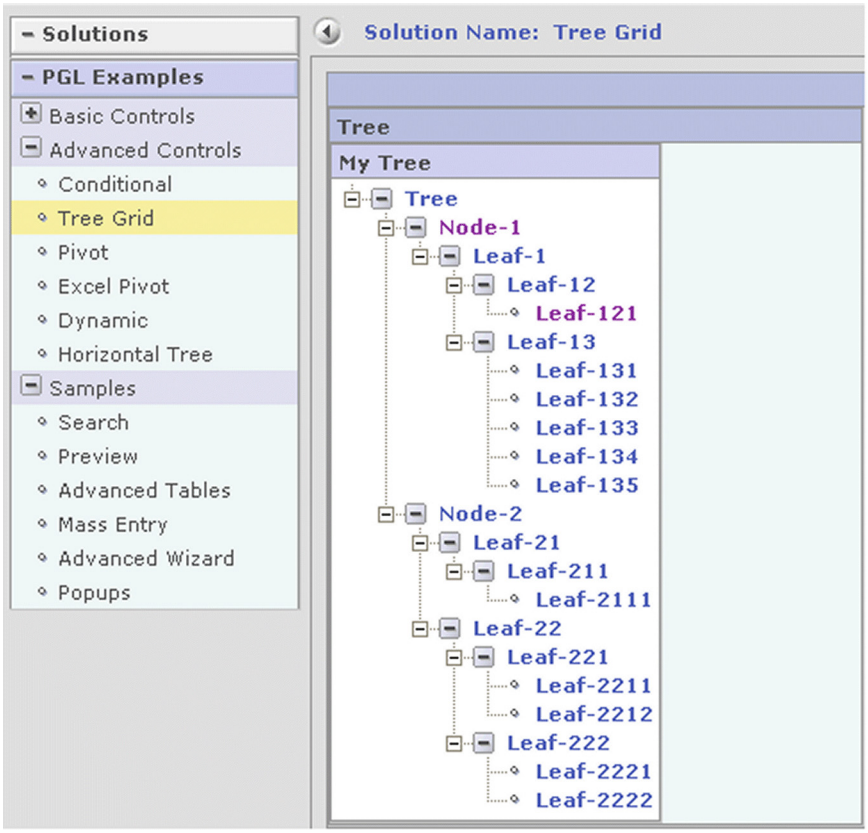
**Workflow**: AdvancedControls

**Step**: Pivot

**PreProcessing:**

```
<TO_DOCVAR AssignToVar="inXml">
        <SAMPLE_XML>
          <RESPONSE>
            <Specification DisplayText="Tree">
              <L0 ID="L0" DisplayText="Node-1">
                <L1 ID="L1" DisplayText="Leaf-1">
                  <L2 ID="L2" DisplayText="Leaf-12">
                    <L3 ID="L3" DisplayText="Leaf-121"/>
                  </L2>
                  <L4 ID="L4" DisplayText="Leaf-13">
                    <L5 ID="L5" DisplayText="Leaf-131"/>
                    <L6 ID="L6" DisplayText="Leaf-132"/>
                    <L7 ID="L7" DisplayText="Leaf-133"/>
                    <L8 ID="L8" DisplayText="Leaf-134"/>
                    <L9 ID="L9" DisplayText="Leaf-135"/>
                  </L4>
                </L1>
              </L0>
              <L10 ID="L10" DisplayText="Node-2">
                <L11 ID="L11" DisplayText="Leaf-21">
                  <L12 ID="L12" DisplayText="Leaf-211">
                    <L13 ID="L13" DisplayText="Leaf-2111"/>
                  </L12>
                </L11>
                <L14 ID="L14" DisplayText="Leaf-22">
                  <L15 ID="L15" DisplayText="Leaf-221">
                    <L16 ID="L16" DisplayText="Leaf-2211"/>
                    <L17 ID="L17" DisplayText="Leaf-2212"/>
                  </L15>
                  <L18 ID="L18" DisplayText="Leaf-222">
                    <L19 ID="L19" DisplayText="Leaf-2221"/>
                    <L20 ID="L20" DisplayText="Leaf-2222"/>
                  </L18>
                </L14>
              </L10>
            </Specification>
          </RESPONSE>
        </SAMPLE_XML>
</TO_DOCVAR>
```

Figure 70: Vertical Tree Example




## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 83: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| checkBoxEnabled | Default is 'false'. If it is set to true, it puts checkboxes and uses the "ID" of the node for the same (if you want to multi-select items from the tree). |
| collapsable | Set to true if container can be collapsable. Default value is false. |
| collapseImage | URL link to the location of the collapse image. |
| displayCondition | Conditional that determines if the field or column should be displayed or not. |

Table 83: Optional Attributes

| Attribute | Description |
|---|---|
| displayText | Text to be displayed as the title of the pad. Automatically internationalized. |
| expandImage | URL link to the location of the expand image. |
| height | Default is '400'. |
| id | Default is 'id1'. |
| leafImage | URL link to the location of the leaf image. |
| name | Specifies the name of the parameter that is submitted as part of the form. |
| postAction | If set to true the action on click will be in postAction. If not the action on click is in preAction |
| scrollable | Set to true if container can be scrollable. Necessary javascript is automatically included. Default value is false. |
| select | An XPath expression used for culling the data for the component's children. The value is automatically evaluated as an expression. |
| width | Default is '100'. |

### Used By

ui:step

### Content

ui:tree-node

# Workbook

The workbook is like "Excel-workbook". A workbook can have more than one spreadsheet and each spreadsheet can have more than one row (usually driven by data) and each row has more than one field (usually driven by data). The spreadsheet is an extension of "table" and the workbook is a container to host multiple "spreadsheets".

**Note:** For <ui:workbook> tag to work, install Microsoft Office Web Components for Office XP (owc10.exe).

### Syntax

The following is the syntax for Workbook.

```
<ui:workbook>
```

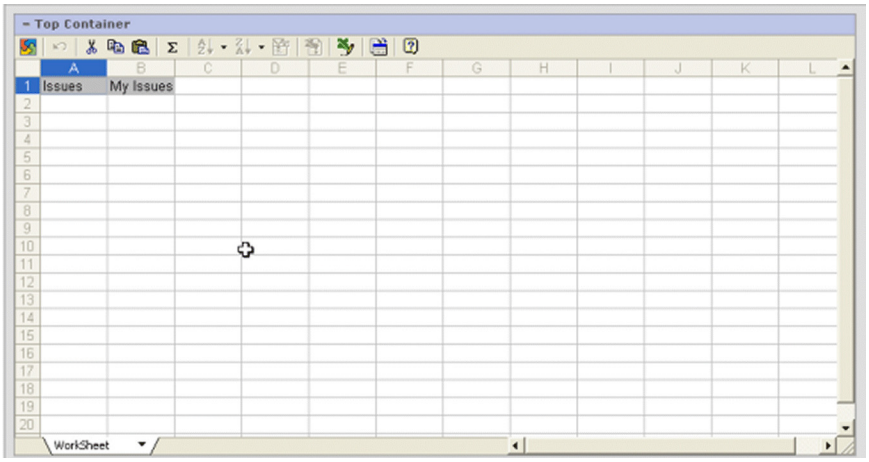
## Example Data

### Example 1:

### PGL:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_form" source="studio.controller:load">
  <ui:container id="container" displayText="Top Container"
collapsable="true" scrollable="true">
    <ui:step displayText="SpreadSheet Example1" id="top_container"
editable="true" scrollable="true">
      <ui:workbook id="ss1" name="ss1" height="400" width="100%"
select="$root">
        <ui:spreadsheet sheetName="WorkSheet" displayText="Existing
Notes" select="NOTES" protect="true">
          <ui:spreadsheet-row select="NOTE">
            <ui:spreadsheet-field name="RP2" displayText="Issues"/>
            <ui:spreadsheet-field name="RP1" displayText="My Issues"/>
          </ui:spreadsheet-row>
        </ui:spreadsheet>
      </ui:workBook>
    </ui:step>
  </ui:container>
</ui:page-group>
```

Figure 71: Work Book Example




### Example 2:

### PGL:

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_form" source="studio.controller:load">
  <ui:container id="container" displayText="Top Container"
collapsable="true" scrollable="true">
    <ui:step displayText="Top Container" id="top_container"
editable="true" scrollable="true">
      <ui:validation/>
      <!-- bootstrap fields -->
      <!-- these should even be hidden from templates -->
      <ui:field-group select="$root">
        <ui:hidden-field name="EV_NAME"/>
```

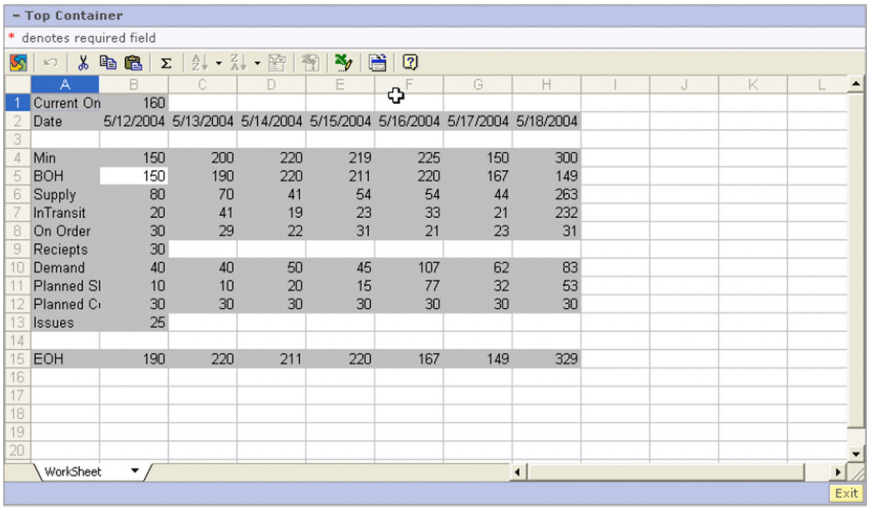
```
        <ui:hidden-field name="EV_KEY"/>
        <ui:hidden-field name="CURRENT_USER_ID" value="{identity-user-
id()}"/>
      </ui:field-group>
      <!-- bootstrap fields done-->
      <ui:workbook id="ss1" name="ss1" height="400" width="100%"
select="$root">
        <ui:spreadsheet sheetName="WorkSheet" displayText="Existing
Notes" select="NOTES" protect="true">
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Current OnHand"/>
            <ui:spreadsheet-field value="160" type="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Date"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,1)+1" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,2)+1" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,3)+1" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,4)+1" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,5)+1" dataType="DateTime"/>
            <ui:spreadsheet-field cellName="StartDateList" value="2004-
05-12" formula="=INDEX(StartDateList,1,6)+1" dataType="DateTime"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row/>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Min"/>
            <ui:spreadsheet-field name="Min1" cellName="MinList"
value="150" dataType="Number"/>
            <ui:spreadsheet-field name="Min2" cellName="MinList"
value="200" dataType="Number"/>
            <ui:spreadsheet-field name="Min3" cellName="MinList"
value="220" dataType="Number"/>
            <ui:spreadsheet-field name="Min4" cellName="MinList"
value="219" dataType="Number"/>
            <ui:spreadsheet-field name="Min5" cellName="MinList"
value="225" dataType="Number"/>
            <ui:spreadsheet-field name="Min6" cellName="MinList"
value="150" dataType="Number"/>
            <ui:spreadsheet-field name="Min7" cellName="MinList"
value="300" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="BOH"/>
            <ui:spreadsheet-field cellName="BOHList" value="150"
dataType="Number" editable="true"/>
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,1)" dataType="Number"/>
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,2)" dataType="Number"/>
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,3)" dataType="Number"/>
```

```
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,4)" dataType="Number"/>
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,5)" dataType="Number"/>
            <ui:spreadsheet-field cellName="BOHList"
formula="=INDEX(EOHList,1,6)" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Supply"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,1)+INDEX(OnOrderList,1,1)+INDEX(Receipts
List,1,1)" dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,2)+INDEX(OnOrderList,1,2)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,3)+INDEX(OnOrderList,1,3)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,4)+INDEX(OnOrderList,1,4)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,5)+INDEX(OnOrderList,1,5)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,6)+INDEX(OnOrderList,1,6)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="SupplyList"
formula="=INDEX(IntransitList,1,7)+INDEX(OnOrderList,1,7)"
dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="InTransit"/>
            <ui:spreadsheet-field name="Min1" cellName="IntransitList"
value="20" dataType="Number"/>
            <ui:spreadsheet-field name="Min2" cellName="IntransitList"
value="41" dataType="Number"/>
            <ui:spreadsheet-field name="Min3" cellName="IntransitList"
value="19" dataType="Number"/>
            <ui:spreadsheet-field name="Min4" cellName="IntransitList"
value="23" dataType="Number"/>
            <ui:spreadsheet-field name="Min5" cellName="IntransitList"
value="33" dataType="Number"/>
            <ui:spreadsheet-field name="Min6" cellName="IntransitList"
value="21" dataType="Number"/>
            <ui:spreadsheet-field name="Min7" cellName="IntransitList"
value="232" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="On Order"/>
            <ui:spreadsheet-field name="Min1" cellName="OnOrderList"
value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min2" cellName="OnOrderList"
value="29" dataType="Number"/>
            <ui:spreadsheet-field name="Min3" cellName="OnOrderList"
value="22" dataType="Number"/>
            <ui:spreadsheet-field name="Min4" cellName="OnOrderList"
value="31" dataType="Number"/>
```

```
            <ui:spreadsheet-field name="Min5" cellName="OnOrderList"
value="21" dataType="Number"/>
            <ui:spreadsheet-field name="Min6" cellName="OnOrderList"
value="23" dataType="Number"/>
            <ui:spreadsheet-field name="Min7" cellName="OnOrderList"
value="31" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Reciepts"/>
            <ui:spreadsheet-field name="RP1" cellName="ReceiptsList"
value="30" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Demand"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,1)+MAX(INDEX(PlannedConsumptionList,1,
1), INDEX(IssuesList,1,1))" dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,2)+INDEX(PlannedConsumptionList,1,2)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,3)+INDEX(PlannedConsumptionList,1,3)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,4)+INDEX(PlannedConsumptionList,1,4)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,5)+INDEX(PlannedConsumptionList,1,5)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,6)+INDEX(PlannedConsumptionList,1,6)"
dataType="Number"/>
            <ui:spreadsheet-field cellName="DemandList"
formula="=INDEX(PlannedShipList,1,7)+INDEX(PlannedConsumptionList,1,7)"
dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Planned Ship"/>
            <ui:spreadsheet-field name="Min1" cellName="PlannedShipList"
value="10" dataType="Number"/>
            <ui:spreadsheet-field name="Min2" cellName="PlannedShipList"
value="10" dataType="Number"/>
            <ui:spreadsheet-field name="Min3" cellName="PlannedShipList"
value="20" dataType="Number"/>
            <ui:spreadsheet-field name="Min4" cellName="PlannedShipList"
value="15" dataType="Number"/>
            <ui:spreadsheet-field name="Min5" cellName="PlannedShipList"
value="77" dataType="Number"/>
            <ui:spreadsheet-field name="Min6" cellName="PlannedShipList"
value="32" dataType="Number"/>
            <ui:spreadsheet-field name="Min7" cellName="PlannedShipList"
value="53" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Planned Consumption"/>
            <ui:spreadsheet-field name="Min1"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min2"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
```

```
            <ui:spreadsheet-field name="Min3"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min4"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min5"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min6"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
            <ui:spreadsheet-field name="Min7"
cellName="PlannedConsumptionList" value="30" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="Issues"/>
            <ui:spreadsheet-field name="RP1" cellName="IssuesList"
value="25" dataType="Number"/>
          </ui:spreadsheet-row>
          <ui:spreadsheet-row/>
          <ui:spreadsheet-row>
            <ui:spreadsheet-field displayText="EOH"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,1)+INDEX(SupplyList,1,1)-
INDEX(DemandList,1,1)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,2)+INDEX(SupplyList,1,2)-
INDEX(DemandList,1,2)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,3)+INDEX(SupplyList,1,3)-
INDEX(DemandList,1,3)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,4)+INDEX(SupplyList,1,4)-
INDEX(DemandList,1,4)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,5)+INDEX(SupplyList,1,5)-
INDEX(DemandList,1,5)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,6)+INDEX(SupplyList,1,6)-
INDEX(DemandList,1,6)" dataType="Number"/>
            <ui:spreadsheet-field cellName="EOHList"
formula="=INDEX(BOHList,1,7)+INDEX(SupplyList,1,7)-
INDEX(DemandList,1,7)" dataType="Number"/>
          </ui:spreadsheet-row>
        </ui:spreadsheet>
      </ui:workBook>
      <ui:buttons id="buttons">
        <ui:button id="SUBMIT" name="SUBMIT" displayText="Exit"/>
      </ui:buttons>
    </ui:step>
  </ui:container>
</ui:page-group>
```

Figure 72: WorkBook Example




**Example 3:**

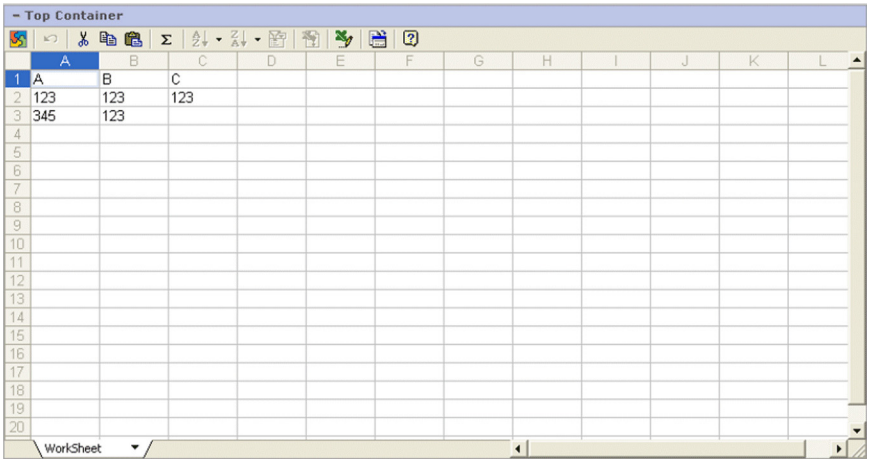
**PGL:**

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_form" source="studio.controller:load">
  <ui:container id="container" displayText="Top Container"
collapsable="true" scrollable="true">
    <ui:step displayText="Top Container" id="top_container"
editable="true" scrollable="true">
      <ui:workBook id="ss1" name="ss1" height="400" width="100%"
select="$root">
        <ui:spreadsheet sheetName="WorkSheet" displayText="Existing
Notes" select="NOTES" protect="true" rowSrc="NOTES">
        </ui:spreadsheet>
      </ui:workBook>
    </ui:step>
  </ui:container>
</ui:page-group>
```

**Data:**

```
<action>
      <TO_DOCVAR AssignToVar="inXml">
         <SAMPLE_XML>
         <NOTES>
         <NOTES>
            <A Value="123"/>
            <B Value="123"/>
            <C Value="123"/>
         </NOTES>
               <NOTES>
            <A Value="345"/>
            <C Value="123"/>
         </NOTES>
      </NOTES>
      </SAMPLE_XML>
      </TO_DOCVAR>
</action>
```

Figure 73: WorkBook Example




## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 84: Optional Attributes

| Attribute | Description |
|---|---|
| activity | Conditional that determines if the component should be displayed or not based on the user having permission for at least one of the listed activities. |
| displayCondition | Conditional that determines if the component should be displayed or not. |
| displayText | Text to be displayed as the label or header of the field. Automatically internationalized. |
| height | Default is '400'. |
| id | Default is 'id1'. |
| name | Specifies the name of the parameter that is submitted as part of the form. |
| width | Default is '100'. |

## Used By

ui:page-group, ui:step

## Content

ui:spreadsheet

# Shell

## Syntax

The following is the syntax for Shell.

```
<ui:shell>
```

## Example Data

Data is not required for this component.

## Required Attributes

None.

## Optional Attributes

The following are the optional attributes:

Table 85: Optional Attributes

| Attribute | Description |
|---|---|
| about-action | The URL for the about box. |
| app-url | The URL for the appFrame panel. |
| background | Sets the background image to use. |
| content-url | The URL for the content frame. This contains the navigation and appFrame. |
| forced-logout | Set to true if the specified logout action should be used to confirm with the user. Default value is false. |
| framed | Set to false if frames are not to be used. Default value is true. |
| help-action | The URL for the global help page. |
| logo | Sets the logo image to use on top left of page. |
| logout-action | The URL for the logout button. |
| nav-url | The URL for the navigation panel. |
| user-name | The user name to display on the top right corner. |

## Used By

ui:page-group

## Content

None.

# DIV

The <UI:DIV> tag defines a division or a section in an HTML document. The <UI:DIV> tag is used to group block-elements to format them with styles.

## Syntax

The following is the syntax for <UI:DIV> tag.

```
<ui:div id="ID_FOR_DIV_TAG" style="width: 800px; height: 400px;"/>
```

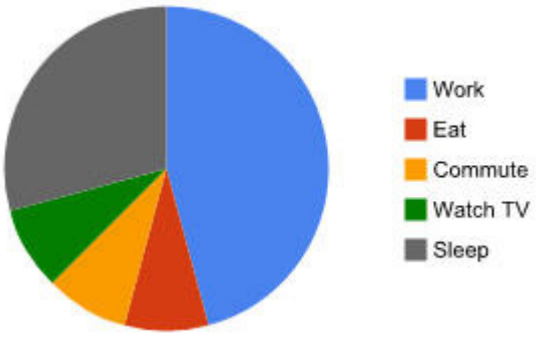
## Example

The following is an example of using <UI:DIV> tag.

```
<ui:page-group xmlns:ui="http://www.teradata.com/ui"
name="studio.templates.simple_entry" source="studio.controller:load">
  <ui:container id="container">
    <ui:step id="top_container" displayText="Summary Screen">
      <ui:script id="script1" src="http://www.google.com/jsapi" />
      <ui:script id="script2">google.load('visualization', '1',
{packages: ['piechart']});
            function drawVisualization()
        {
               // Create and populate the data table.
               var data = new google.visualization.DataTable();
               data.addColumn('string', 'Task');
               data.addColumn('number', 'Hours per Day');
               data.addRows(5);
               data.setValue(0, 0, 'Work');
               data.setValue(0, 1, 11);
               data.setValue(1, 0, 'Eat');
               data.setValue(1, 1, 2);
               data.setValue(2, 0, 'Commute');
               data.setValue(2, 1, 2);
               data.setValue(3, 0, 'Watch TV');
               data.setValue(3, 1, 2);
               data.setValue(4, 0, 'Sleep');
               data.setValue(4, 1, 7);

               // Create and draw the visualization.
               new
google.visualization.PieChart(document.getElementById('visualization')).
               draw(data, null);
            }
            google.setOnLoadCallback(drawVisualization);</ui:script>
      <ui:div id="visualization" style="width: 300px; height: 300px;" />
    </ui:step>
  </ui:container>
</ui:page-group>
```

The following figure illustrates the preceding code.

Figure 74: UI:DIV




## Example Data

Data is not required for this component.

## Required Attributes

The following table lists the required attributes for UI:DIV tag.

Table 86: Required Attributes

| Attribute | Description |
|---|---|
| id | Identifier for HTML component. |

## Optional Attributes

The following table lists the optional attributes for UI:DIV tag.

Table 87: Optional Attributes

| Attribute | Description |
|---|---|
| Style | Specify a style controlling the appearance of cell's contents. |

## Used By

ui:step

## Content

None.

# CHAPTER 6 MDM Elements

### What's In This Chapter

This chapter gives you an introduction to Master Data Management (MDM) Elements.

Topics include:


- Master Data Management
- MDM Elements


# Master Data Management

TD MDM Platform framework separates process from data, enabling Information Technology (IT) organizations to meet their business needs.

As business practices evolve, and companies are forced to rapidly streamline their processes to meet consumer demands, the biggest challenge faced by them is the inability of the IT systems to keep pace. Most IT organizations lack the ability to quickly adopt new business models. Every business process is supported by multiple applications, each having its own data model and workflows. Often, custom solutions and point-to-point bridges have to be built so that data and transaction integrity across these multiple applications can be maintained. However, they are inflexible to change and cost prohibitively high to maintain as the business processes go through subsequent rounds of evolution.

The key to MDM is its ability to enable flexible business process management using a Services Based Architecture (SBA). It provides a distributed transaction framework and robust integration framework to enable flawless execution that scales as transaction volumes increase. It has a powerful business programming environment to describe business rules, workflows, and data models and is based on open-standards (XML, Web-Services).

In an abstract sense, a business workflow involves moving a document through a series of steps; at each step, more information is added to the document. The key to this framework is its ability to enable intuitive definition and extension of workflows with very little or no programming. For the workflow steps that require user-intervention, MDM offers a powerful interface for building the UI screens and the related plumbing for data input and output.

## MDM Studio

An integrated development environment (IDE) for designing, debugging and maintaining the workflows, data models, and user-interfaces that together define a business process. Web-Services, used to define the integration touch points with external applications, can be imported inside Studio and used seamlessly within workflows.

### Run-Time Server

The workflows, data-models and user-interfaces designed in MDM Studio are deployed in a production environment using the Run-Time server. It scales as the transaction volumes increase and can easily be configured for load-balancing, fail-over etc.

One of the key goals of MDM is to avoid building point solutions for every new workflow. The framework promotes reuse of components like business rules, UI screens, documents, sub-workflows, etc. intuitively, thus enabling users to define, and later extend, new workflows with ease.

# MDM Elements

The main components of the MDM framework that are used in any application are:

- X-Documents: Describes the data model or documents in the system.
- X-Rules: Defines the business rules in relation to the X-Documents. Analogous to functions in conventional programming languages.
- X-Workflows: Model the business processes (document flows through the system together with rules, events, timers etc.).
- X-Services: Group of X-Workflows that make up a logical unit or a business contract.
- X-Server: A single JVM (Java Virtual Machine) instance that implements one or more X-Services. It constitutes the Run-Time environment for MDM.

All these components are defined and/or configured using the MDM Studio; refer to the *MDM Platform Studio User Guide* to learn the various tools and techniques involved in the process.

### X-Server

The X-Server is an instance of the MDM application, running in a single JVM (Java Virtual Machine). It is the Run-Time deployment of the application that was designed using the MDM Studio.

The main features of X-Server are:

- Multiple X-Services can be run within an X-Server (configurable at design time).
- Multiple X-Servers can be run on a single machine on different ports.
- Communicates with other X-Servers through a variety of protocols - XRMP, TCPA, HTTP.
- It is a stateless server.
- One X-server connects to only one RDBMS.
- It can run in a stand-alone mode (single X-server) or a distributed mode (multiple X-servers) to guarantee high scalability and reliability.

## X-Service

An X-Service is a component that satisfies a business contract; analogous to a business function. It contains a group of workflows that together form a logical unit.

The main features of X-Service are:

- The business contracts are specified through XML API.
- All services are stateless.
- Typically all the services that make up an application would be deployed in a single X-Server.
- All the X-Services are registered with the Locator service. Clients can find the services by querying the Locator service.

The locator Service is a core MDM framework service that is used in every implementation. It maintains a registry of all the X-Service names and the X-Server URLs which form the MDM application.

The main features of Locator Service are:

- Provides API for registering and un-registering services.
- Allows clients to discover the location of the services by querying the locator for a service and getting the URL of the server on which it is deployed.
- If a service is deployed by more than one server, then it performs load balancing by giving out the server URLs in a round-robin scheme.
- The client communication libraries hide the nuances of finding the service through the locator and have the built-in intelligence to switch over to another server (by contacting the locator) in case of a failure.

## X-Workflow

X-Workflows are an XML based specification to model business processes and embed execution logic.

The main features of X-Workflow are:

- MDM provides built-in workflow nodes for:
  - Marking the start and end of workflows (Start, Done nodes)
  - Performing actions (Task node)
  - Creating user-interfaces (UI node)
  - Wait for events (Event node)
  - Controlling the execution (Decision, And, Or, Branch, and Sub-Workflow nodes)
  - Wait for a specified time (Timer node)
  - Specialized actions like document validation, messaging, and so on.
- Design allows for creation of new workflow nodes to perform specialized tasks.
- Can be re-used by modeling them as sub-workflows. The sub-workflows can then be composed to create complete complex process flows.
- Can be instantiated explicitly or through an event.

## X-Rules

X-Rules are an XML specification for expressing business logic in workflows. Each X-Rule has a set of conditions and a set of actions; the actions are executed only if the conditions are satisfied.

The main features of X-Rules are:

- Ability to perform most of the workflow actions by selecting actions from the built-in action list provided by the X-Rule editor.
- Built-in action primitives for common X-Document operations - analogous to data manipulation language statements.
- Tight synergy between X-Rules and X-documents makes it natural to execute actions on documents in X-Rules.
- X-Path semantics can be applied to traverse and manipulate X-document instances within X-Rules. X-Path expression functions are extensible and customizable.
- Hooks to perform complex logic processing in Java.
- Design allows for definition of new XML action semantics to perform tasks not available in the core product.

## X-Documents

An XML specification to model the business document in the MDM framework; consists of document name, properties, and data types, which provides the basis for database table definitions.

The main features of X-Documents are:

- Impart transparency to underlying RDBMS since the XML specification is not database-specific.
- Contain relationships to other X-Documents, which provide the basis for database schema referential and integrity constraints.
- Supports 1-1, 1-n, n-n relationships with other X-Documents.
- Extensible at deployment time to include additional properties and relationships.
- Audit Trail and Dynamic Attributes can be turned on at deployment.
- Provides querying and persistent business document instances - XML based data manipulation statements that are translated to database dependent SQL statements.

# SECTION B Service Reference

Section B provides link to the various chapters on Service Reference.


- Chapter 7: “X-Document.”
- Chapter 8: “x-Rules.”
- Chapter 9: “X-Operations.”
- Chapter 10: “X-Path.”
- Chapter 11: “MDM JSON Framework.”
- Appendix C: “Timer Service.”
- Appendix D: “Validation Services.”
- Appendix E: “Upload Services.”
- Appendix F: “ID Generation.”
- Appendix G: “Events and Request Interfaces.”
- Appendix H: “Using Rules Language Project.”
- Appendix I: “X-Service APIs and WSDLs.”
- Appendix J: “Data Relationship Management APIs.”
- Appendix K: “MDM Generic APIs.”
- Appendix L: “Non MDM APIs.”
- Appendix M: “DB Based Locking Utility.”
- Appendix N: “Export Framework.”
- Appendix O: “Purge Framework.”
- Appendix P: “Search Framework.”
- Appendix Q: “State Machine.”
- Appendix R: “Workflow Specification Framework.”
- Appendix S: “Messaging and E-mail Service.”
- Appendix T: “Test Service.”
- Appendix U: “MDM_SYS_ENT_STATE.”
- Appendix V: “Multiple Table Updates using DocView.”
- Appendix W: “Parent-Child Validation.”
- Appendix X: “Dashboard Technology Implementation.”
- Appendix AA: “Frequently Asked Questions.”

# CHAPTER 7 X-Document

### What's In This Chapter

This chapter gives you information about X-Document.

Topics include:


- Introduction
- Document Identifiers
- Surrogate Key
- Document Index
- Document Properties
- Document Links
- Foreign Key Constraints
- Overlay Documents
- Audit Trail
- Document Constraints


## Introduction

X-Document is a XML specification to model application data and their relationships. An X-Document has the following:

- Name of the document.
- Attributes that the document can contain and the related data types. These attributes are called Document Properties.
- The links (or relationships), the document has with other documents. These links are called Document Links.

The following is an example of an X-Document.

```
<DOCUMENT Name=”CustomerOrder” TableName="MST_CUSTOMER_ORDER"
SurrogateKey="true" AuditTrail="true">

 <PROPERTY Name="CreatedBy" Type="string(50)"
ColumnName="CREATED_BY" Required="true"/>

<PROPERTY Name="CreatedDate" Type="date" ColumnName="CREATED_DATE"
Required="true"/>
```

```
<PROPERTY Name="ID" Type="string(50)" PrimaryKey="true"
Required="true"/>

<PROPERTY Name="TotalPrice" Type="float" ColumnName="TOTAL_PRICE"
Required="true"/>

<DOCUMENTLINK Name="CO_CREATED_BY" LinkedDocument="User">

<MAP_PROPERTY From="CreatedBy" To="ID"/>

</DOCUMENTLINK>

</DOCUMENT>
```

In the above example,

- `CUSTOMER_ORDER` is the name of the document.
- `CreatedBy`, `CreatedDate`, `ID`, `TotalPrice` are the properties of the document.
- The `CUSTOMER_ORDER` document has a relationship with `USER` document through the link `CO_CREATED_BY`.

The following is an example of an instance of a CUSTOMER_ORDER document.

```
<CUSTOMER_ORDER>

<ID Value ="CO_1490" />

<CreatedDate Value ="04/20/2004"/>

<CreatedBy Value ="John Doe"/>

<TotalPrice Value =" 7000.0 "/>

</CUSTOMER_ORDER>
```

The fields, whose values are `NULL`, are not contained in the document instance.

**Note:** An X-Document is similar to a table definition in a RDBMS and the X-Document instance is like a row in the RDBMS table.

# Document Identifiers

The X-Document properties that are declared as Primary Key uniquely identify the document.

For example, ID is specified as a primary key in the above example. Therefore, specifying the ID uniquely identifies the `CUSTOMER_ORDER` document. More than one document property can be declared as a primary key.

The following is an example of a X-Document with more than one primary key.

```
<DOCUMENT Name="LINE_ITEM">

<PROPERTY Name ="ITEM_ID" Type ="string(40)"

PrimaryKey ="yes"/>

<PROPERTY Name ="PO_ID" Type ="string(40)" PrimaryKey ="yes"/>
```

```
<PROPERTY Name="CREATION_DATE" Type ="date"/>
<PROPERTY Name="STATE" Type ="string"/>
<PROPERTY Name="QUANTITY" Type ="int"/>
<PROPERTY Name="INVOICE" Type ="float"/>
<PROPERTY Name="SELLER " Type ="string"/>
<DOCUMENTLINK Name="PO_INFO" LinkedDocument= "PURCHASE_ORDER">
<MAP_PROPERTY To="ID" From="PO_ID"/>
</DOCUMENTLINK>
<DOCUMENTLINK Name="SELLER_INFO" LinkedDocument="PROFILE">
<MAP_PROPERTY To="ID" From="SELLER_ID"/>
</DOCUMENTLINK>
</DOCUMENT>
```

In the above example, `ITEM_ID` and `PO_ID` properties uniquely identify a `LINE_ITEM` document.

A document can have more than one set of properties that make it unique. That is, a document can have one or more alternate primary keys. You can specify alternate primary keys by using the `UNIQUE` tags.

The following is an example of a X-Document with one or more alternate primarykeys.

```
<DOCUMENT Name="PROFILE"/>
<PROPERTY Name="ID" Type ="string(40)" PrimaryKey="yes"/>
<PROPERTY Name="FIRST_NAME" Type ="string(40)"/>
<PROPERTY Name="LAST_NAME" Type ="string(40)"/>
<PROPERTY Name="DESIGNATION" Type ="string"/>
<PROPERTY Name="PHONE" Type ="string"/>
<PROPERTY Name="SSN" Type ="string"/>
<PROPERTY Name="JOB_LEVEL " Type ="int"/>
<UNIQUE>
<KEY Property="FIRST_NAME"/>
<KEY Property="LAST_NAME"/>
</UNIQUE>
<UNIQUE>
<KEY property="SSN"/>
</UNIQUE>
```

```
</DOCUMENT>
```

In the above example, the sets `{ID}`, `{FIRST_NAME, LAST_NAME}`, and `{SSN }` uniquely identify the `PROFILE` documents.

# Surrogate Key

Documents with only one Primary Key can specify SurrogateKey as true. When this is specified, MDM takes care of generating the value for the Primary Key column for any new rows added using the `ADD_DOCUMENT` call. Users can also optionally specify an IdPrefix attribute which can be used in conjunction to generate surrogate keys.

For example:

```
<DOCUMENT Name="PARENT" SurrogateKey="true" IdPrefix="PT-">
<PROPERTY Name="ID" Type="string(32)" PrimaryKey="yes"/>
<PROPERTY Name="NAME" Type="string(32)" Required="yes"/>
<PROPERTY Name="DATA" Type="LongString"/>
</DOCUMENT>
```

Upon an `ADD_DOCUMENT` invocation, MDM will generate a value for the Primary Key column `ID` if one hasn't been specified already. `IdPrefix` will be used as the prefix for the generated value. In this case, the `ID` value could be something like PT-73258.

# Document Index

You can set up indexes on the properties that are frequently used in the query criteria. You can speed up querying the documents from the database using indexes. You can set up multiple indexes on a document. An index can have more than one property as its key.

However, note that using indexes will slow down inserting new rows to the table because the database re-computes indexes whenever a new row has to be inserted.

The following is an X-Document with indexes.

```
<DOCUMENT Name="LINE_ITEM">
<PROPERTY Name="ITEM_ID" Type="string(40)" PrimaryKey= "yes"/>
<PROPERTY Name="PO_ID" Type = "string(40)" PrimaryKey = "yes"/>
<PROPERTY Name="CREATION_DATE" Type="date"/>
<PROPERTY Name="STATE" Type="string"/>
<PROPERTY Name="QUANTITY" Type="int"/>
<PROPERTY Name="INVOICE" Type="float"/>
```

```
<PROPERTY Name="SELLER " Type="string"/>
<PROPERTY Name="EXT_ID " Type="string"/>
<PROPERTY Name="ORDER_TYPE" Type="string"/>
<DOCUMENTLINK Name="PO_INFO" LinkedDocument= "PURCHASE_ORDER">
<MAP_PROPERTY To="ID" From="PO_ID"/>
</DOCUMENTLINK>
<DOCUMENTLINK Name="SELLER_INFO" LinkedDocument="PROFILE">
<MAP_PROPERTY To="ID" From="SELLER_ID"/>
</DOCUMENTLINK>
<INDEX>
<KEY Property="PO_ID"/>
</INDEX>
<INDEX>
<KEY Property="EXT_ID"/>
<KEY Property="ORDER_TYPE"/>
</INDEX>
</DOCUMENT>
```

In the above example, queries providing values for `{EXT_ID, ORDER_TYPE}` or `{PO_ID}` will be faster because of the indexes that have been set up.

# Document Properties

The following are the valid data types for a document property:

- boolean
- char
- clob
- date
- decimal
- double
- encryptedString
- float
- int
- String
- timestamp
- xml

## boolean

*Syntax:*

```
<PROPERTY Name="mandatory" Type="boolean" ColumnName="MANDATORY"
Default="false">
      <VALID_VALUES>
        <OPTION Value="true" />
        <OPTION Value="false" />
      </VALID_VALUES>
 </PROPERTY>
```

## char

*Syntax:*

```
<PROPERTY Name="SEVERITY" Type="Char"/>
```

## clob

*Syntax:*

```
<PROPERTY Name="RULE_EXPR" Type="Clob" ColumnName="RULE_EXPR" />
```

*Description:*

The `clob` data type can be used to store a block of text. It is designed to store ASCII text data, including formatted text such as HTML or PostScript.

## date

*Syntax:*

```
<PROPERTY Name="CREATION_DATE" Type="date"/>
```

## decimal

*Syntax:*

```
<PROPERTY Name="INVOICE_VALUE" Type="Decimal"/>
```

## double

*Syntax:*

```
<PROPERTY Name="QuantityOnHand" Type="double(15, 4)"
ColumnName="CAD_QTY_ON_HAND" />
```

## encryptedString

*Syntax*:

```
<PROPERTY Name="PASSWORD" Type="encryptedString(16)"/>
```

*Description*:

The encryptedString property is used to represent sensitive information like password. You can encrypt the field prior to storing it in the database. This field will be decrypted after retrieval. For the application code it is equivalent to simple string since X-Core takes care of encryption or decryption automatically.

Example 1:

```
<PROPERTY Name="PASSWORD" Type="encryptedString(16)"/>
```

## float

*Syntax:*

```
<PROPERTY Name="INVOICE" Type="float"/>
```

## int

*Syntax:*

```
<PROPERTY Name="PIN" Type="int"/>
```

## String

*Syntax*:

```
<PROPERTY Name="STATE" Type="string"/>
```

*Description*:

If you do not specify the length of the string, then a default value of 32 is assigned by X-Core framework.

If you need a string size greater than the default, then you should explicitly specify the size.

Example 1:

```
<PROPERTY Name="STATE" Type="string"/>
```

In this example the property is assigned a string data type whose maximum size can be 32

Example 2:

```
<PROPERTY Name="STATE" Type="string(40)"/>
```

In this example the property is assigned a string data type whose maximum size can be 40.

## ST_Geometry

*Syntax*:

```
<PROPERTY Name="LOCATION" Type="ST_Geometry"/>
```

*Description*:

ST_Geometry represents any of the following geometry types: ST_Point, ST_LineString, ST_Polygon, ST_GeomCollection, ST_MultiPoint, ST_MultiLineString, ST_MultiPolygon, and GeoSequence.

**Note:** ST_Geometry data type is supported only on Teradata 13.10.00.00 and above.

### timestamp

*Syntax:*

```
<PROPERTY Name="LAST_MODIFIED" Type="timestamp"/>
```

*Description:*

The value of the timestamp property is always output as `mm/dd/yyyy hh:MM:ss SSS`. For example, 04/23/2001 15:30:45 345 is the same as April 23, 2001 3:30 PM and 45 seconds and 345 milliseconds format.

The date, datetime and timestamp differ only in the precision

### xml

*Syntax:*

```
<PROPERTY Name="DESC" Type="xml"/>
```

# Document Links

A document link is similar to a hyperlink in a HTML document. The primary motivation of a document link is to navigate the additional information associated with the document.

The following is a scenario for defining the link to the `PURCHASE_ORDER` document:

- When you are displaying the details of a `LINE_ITEM` in a UI, you might want to provide some information on the `PURCHASE_ORDER`. It is desirable to have the `PURCHASE_ORDER` details served up when you are querying for the `LINE_ITEM`.
- A document link specifies the following:
- The name of the document that is being linked.
- The properties of the current document (From) that are mapped on to the linked document (To).
- A logical name for the document link.

Example 1:

```
<DOCUMENT Name="LINE_ITEM ">

.................

.................

<DOCUMENTLINK Name="PO_INFO" LinkedDocument= "PURCHASE_ORDER" >

<MAP_PROPERTY To="ID" From="PO_ID"/>
```

```
</DOCUMENTLINK>

</DOCUMENT>
```

In the above example, a document link called `PO_INFO` is created. `PO_INFO` represents a link between the current document that is `LINE_ITEM` and the linked document that is `PURCHASE_ORDER`. The `PO_ID` property of the current document (`LINE_ITEM`) is mapped on to the ID property of linked document (`PURCHASE_ORDER`).

Example 2:

The following is an example of a document link with more than one property

```
<DOCUMENT Name="SHIPMENT ">

<PROPERTY Name="ITEM_ID" Type="string(40)" PrimaryKey="yes"/>

<PROPERTY Name="PURCHASE_ORDER_ID" Type="string(40)"
PrimaryKey="yes"/>

<PROPERTY Name="PROMISED_DATE" Type="string(40)" PrimaryKey="yes"/>

<PROPERTY Name="PROMISED_QTY" Type="string(40)" PrimaryKey="yes"/>

...................

...................

<DOCUMENTLINK Name="LINE_ITEM_INFO" LinkedDocument="LINE_ITEM">

<MAP_PROPERTY To="ITEM_ID" From="ITEM_ID"/>

<MAP_PROPERTY To="PO_ID" From="PURCHASE_ORDER_ID"/>

</DOCUMENTLINK>

</DOCUMENT>
```

Example 3:

The following is an example of multiple link documents attached to the same linked document

```
<DOCUMENT Name="ALERT ">

<PROPERTY Name="ID" Type="string"/>

<PROPERTY Name="FROM_USER" Type="string"/>

<PROPERTY Name="TO_USER" Type="string"/>

<PROPERTY Name="MESSAGE" Type="string(255)"/>

<DOCUMENTLINK Name="FROM_USER_INFO" LinkedDocument="PROFILE">

<MAP_PROPERTY To="ID" From="FROM_USER"/>

</DOCUMENTLINK>

<DOCUMENTLINK Name="TO_USER_INFO" LinkedDocument="PROFILE">

<MAP_PROPERTY To="ID" From="TO_USER"/>
```

```
</DOCUMENTLINK>

</DOCUMENT>
```

In the above example, FROM_USER_INFO and TO_USER_INFO are links to the same linked document, PROFILE.

# Foreign Key Constraints

Whenever you delete a document instance, some linked document instances also need to be deleted.in order to maintain referential integrity. For example, LINE_ITEM document cannot exist without a PURCHASE_ORDER document. Therefore, whenever a PURCHASE_ORDER document instance is deleted, it is necessary to delete the related LINE_ITEM document instance.

Specifying an AutoDelete flag on a document link ensures that all the related document instances are deleted when the document instance is deleted.

Specifying an AutoDelete flag creates a FOREIGN key in the linked document. The properties in the linked document are the FOREIGN keys mapped to the properties in the current document.

The following is an example of a PURCHASE_ORDER document with foreign keys.

```
<DOCUMENT Name="PURCHASE_ORDER" AuditTrail="yes">

.................

.................

<DOCUMENTLINK Name="LINE_ITEMS" LinkedDocument="LINE_ITEM"
AutoDelete="yes">

<MAP_PROPERTY To="PO_ID" From="ID"/>

<DOCUMENTLINK>

.................

.................

</DOCUMENT>
```

In the above example document link, LINE_ITEMS, is flagged with AutoDelete as yes. Whenever the PURCHASE_ORDER is deleted, all linked LINE_ITEM documents are automatically deleted. Specifying the AutoDelete flag results in a FOREIGN_KEY_CONSTRAINT on the LINE_ITEM table in the database. PO_ID property in the LINE_ITEM table is a FOREIGN KEY that references the ID property of the PURCHASE_ORDER table.

# Overlay Documents

Overlay Document enables you to extend an existing document definition to suit the needs of a specific implementation. Overlay documents allows you to define additional inline properties and document links to the existing document definition. You can also turn on audit-trail using this mechanism. The system will apply the additions specified in the overlay document to the existing document definition. Overlay documents cannot re-define any of the properties or document links already specified in the existing document.

Please note that you cannot remove any existing properties or document links using the Overlay document mechanism.

- Important: Inline properties specified in the Overlay document definition are mapped on as additional columns in the table created for the main document. So you have to regenerate the schema whenever you change the Overlay document definition.

Example 1:

The following is the example of defining Overlay Document to `PURCHASE_ORDER` document.

```
<OVERLAY_DOCUMENT Name="PURCHASE_ORDER" AuditTrail="yes">
<PROPERTY Name="EXTERNAL_ID" Type="string(40)"/>
<PROPERTY Name="ORDER_POINT_ID" Type="string(40)"/>
<PROPERTY Name="ORDER_ORIGIN" Type="string(40)"/>
<DOCUMENTLINK Name="ORDER_POINT_INFO" LinkedDocument="PROFILE">
<MAP_PROPERTY To="ID" From="ORDER_POINT_ID"/>
</DOCUMENTLINK>
<PROPERTY Name="EXTERNAL_ID" Type="string(40)">
<UNIQUE>
<KEY Property="EXTERNAL_ID" Type="string(40)"/>
<KEY Property="ORDER_ORIGIN" Type="string(40)"/>
</UNIQUE>
</PROPERTY>
</OVERLAY_DOCUMENT>
```

In the above example, the `PURCHASE_ORDER` document will now have three additional inline properties, have audit trail enabled, have a new document link in addition to the ones defined in the previous example and have a new set of document identifiers.

# Audit Trail

An audit trail enables the tracking of changes made to a document instance. You can construct the document change history from the audit trail information. This feature must be enabled in order to track the changes. This enabling is done by specifying AuditTrail="yes" in the document definition.

The following is an example of an audit trail:

```
<DOCUMENT Name="CAR" AuditTrail="yes">

.....................

.....................

</DOCUMENT>
```

In the above example, the audit trail is enabled by setting the attribute `AuditTrail="yes"`. The system creates a shadow table, `CAR_AUDIT_TRAIL` and logs all the changes that are done on the CAR table, into the shadow table.

The shadow table contains the following fields in addition to those of the CAR table:

- Name of the operation (ADD, MODIFY or DELETE)
- Login Id of the user performing the operation
- Time stamp of the operation

Enabling audit trail on an already existing table using an overlay document will need schema re-generation.

# Document Constraints

Constraints on properties of a document can be specified in the X-Document definition. These constraints will be translated into schema based constraints during schema generation. MDM will NOT use the constraints specified during its execution; it will simply delegate the evaluation of the constraints to the database.

The supported constraints are:

- valid value constraints,
- constraints involving logical operations and operators (including range constraints), and
- literal constraints.

Example1: Valid value constraints.

The property is defined in the xdocument definition as:

```
<PROPERTY Name="STATUS" Type="string(32)" Default="ACTIVE">

<VALID_VALUES>
```

```
<OPTION Value="ACTIVE"/>
<OPTION Value="DORMANT"/>
<OPTION Value="INACTIVE"/>
</VALID_VALUES>
</PROPERTY>
```

Example 2: Constraints with operators

The property is defined in the xdoc document definition as:

```
<PROPERTY Name="RANK" Type="int" ColumnName="RK">
<CONSTRAINT>
<EQUAL Value="10"/>
<OR>
<AND>
<LESS Value="600"/>
<GREATER_EQUAL Value="300"/>
</AND>
<AND>
<LESS_EQUAL Value="700"/>
<GREATER_EQUAL Value="600"/>
</AND>
</OR>
</CONSTRAINT>
</PROPERTY>
```

Example 3: Literal constraints

The property is defined in the xdoc document definition as:

```
<PROPERTY Name="QUANTITY" Type="double" ColumnName="QTY"
Default="30.0">
<CONSTRAINT>
<OR>
<LITERAL Value="2 * #QUANTITY &lt;= 10000.989"/>
<AND>
<GREATER_EQUAL Value="30.0"/>
<LESS_EQUAL Value="5000"/>
</AND>
```

```
</OR>
</CONSTRAINT>
</PROPERTY>

<PROPERTY Name="PRICE" Type="float" Default="2.0">
<CONSTRAINT>
<LITERAL Value="#PRICE * #QUANTITY &gt;= 60.0"/>
</CONSTRAINT>
</PROPERTY>
```

MDM provides a number of built-in operations for manipulating X-Documents. For details, see Chapter 9: “X-Operations”.

# CHAPTER 8 X-Rules

### What's In This Chapter

This chapter provides information about X-Rules and discusses in details the definition, different types, and examples of X-Rules.

Topics include:


- Introduction
- Variables and Expressions
- Business Requests


## Introduction

X-Rules are the XML specification for expressing business logic in workflows and have one of the following two forms:

Form 1:

```
<RULE>
<COND>
{condition statement1}
.............
{condition statementN}
</COND>
<ACTION>
{action statement1}
.............
{action statementN}
</ACTION>
</RULE>
```

Here, each X-Rule has a set of conditions and a set of actions. When an X-Rule is invoked, the actions are executed if ALL the conditions are satisfied.

Form 2:

```
<RULE>
```

```
<ACTION>
{action statement1}
{action statement2}
.............
{action statementN}
</ACTION>
</RULE>
```

Here, the X-Rule does not have any conditions, only actions. Hence, when it is invoked, the actions are executed immediately.

The following example illustrates a simple X-Rule 'helloWorld' that prints the text 'Hello World!' to the console as well returns it to the calling function as an XML value:

```
<DEFINE_METHOD Name="helloWorld">
<RULE>
<ACTION>
<SET Var="myVar" FromValue="Hello World!"/>
<PRINTLN Value="{$myVar}"/>
<TO_DOCVAR AssignToVar="thisReturn">
<RETURN>
<TEXT Value="{$myVar}"/>
</RETURN>
</TO_DOCVAR>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

The above rule is invoked using the `REQUEST` tag as follows:

```
<REQUEST Name="helloWorld" AssignToVar="returnVar"/>
```

The variable '`returnVar`' will be assigned the response value:

```
<RESPONSE Status="Success">
<TEXT Value="Hello World!"/>
</RESPONSE>
```

MDM contains a number of built-in operands that are used to define an X-Rule (see Chapter 9: “X-Operations” for the list of these operations). Furthermore, to perform tasks that are not available in the MDM framework or to re-use external functions, you can:

Define custom operands, i.e. new action tags, easily using the framework extension mechanism.

Use Java to define parts of the business logic and/or leverage existing class libraries, and integrate it seamlessly within the X-Rule using operands such as `NATIVE_FUNCTION` and `INSTANCE_FUNCTION`.

# Variables and Expressions

X-Rules support the following variable types:

- Var: An un-typed variable
- DocVar: A variable of type XML Document
- Value: An un-typed constant
- Property: The property-name of an X-Document
- Attribute: The attribute-name of an XML Document

There are a number of variables that have specific connotation when writing X-Rules. For example, thisParam, thisReturn, thisDoc, and so on. These variables will be covered in later sections, in context with the type of X-Rule that uses them.

Using the above mentioned variable types, you can form complex expressions of the following form:

```
<ActionTag LhsExpression RhsExpression/>
```

OR

```
<ConditionTag LhsExpression RhsExpression/>
```

A LhsExpression is formed using one of the following:

- `Value`
- `SelectList`
- `Var`
- `DocVar and Property`
- `DocVar and Attribute`

A RhsExpression is a LhsExpression with a From prefix:

- `FromValue`
- `FromSelectList`
- `FromVar`

The following examples illustrate how Lhs and Rhs expressions are used with the help of the SET action statement.

Example 1:

The following statement creates an un-typed variable called myVar and assigns it the value of 50.

```
<SET Var="myQuantity" FromValue="50"/>

LhsExpression : Var="myQuantity"

RhsExpression : FromValue="50"
```

## Special Variables

### userName (UI)

This variable is available as pipeline variables in any MDM workflow and captures the login name used by the user to initiate the workflow.

Example 1:

```
<IF_TEST Test="$userName='corporate_user'">

<THEN>

{Perform some actions}

</THEN>

</IF_TEST>
```

The above example tests whether the user who has initiated the workflow is the user with `userName 'corporate_user'`.

### REDIRECT_URL (UI)

This variable is used to set the URL to which the web-page is redirected after a workflow is finished. It is usually set in the DONE node of a workflow.

Example 1:

```
<SET Var="REDIRECT_URL" FromValue="http://MDM.teradata.com"/>
```

Example 2:

```
<SET Var="homePageUrl" FromValue="{concat($webuiContextPath, '/core/
home.jsp')}"/>

<SET_REDIRECT_URL Value="{$homePageUrl}"/>
```

### Service Variables

In an X-Service, when a variable is defined as a service parameter, it is available anywhere within that X-Service (workflows, rules, validations etc.). It can be referenced by simply using the variable name in the RhsExpression as with any other variable (FromVar, FromValue etc.). However, the value of the service variable can only be a constant value and cannot be changed or re-assigned in the workflow run time.

### Global Variables

In addition to the service parameters, which are static in nature, it is possible to declare a variable as a global variable anywhere within a service (workflow, rule etc.) by setting the 'Scope' attribute in the SET operation to 'Global'. This is like setting a 'pipeline variable' which is visible within the entire service. The value in this variable can be overridden by other users, so it will not act as a session variable specific to particular user. Please note that without using the global scope the pipeline variables are visible only within the specific workflow in which they have been defined.

Example:

```
<SET Var="approvalStatus" FromValue="APPROVED" Scope="Global"/>
```

# Condition Statements

A condition statement is any statement that evaluates to a true or false. A conditional statement could be a simple condition or a compound condition. The following are the different types of condition statements

See Simple Condition

See Compound Condition

## Simple Condition

The basic form of the condition statement is:

```
<IS_TRUE LhsExpression />
```

OR

```
<IS_FALSE LhsExpression />
```

Where LhsExpression is an X-Path expression that contains one of the following comparison operators:

- `EQUAL . '='`
- `NOT_EQUAL . '!='`
- `AND . 'and'`
- `OR . 'or'`

In addition, to the above the following operators could also be defined when working with numbers:

- `GREATER . '>'`
- `GREATER_EQUAL . '>='`
- `LESS . '<'`
- `LESS_EQUAL . '<='`

See Boolean X-Path Expressions in X-Path for detailed information on building expressions.

Example 1:

```
<IS_TRUE Value="{int($myDoc/PROMISED_TOTAL_QTY/@Value)=100}"/>
```

The above statement evaluates to true if the value of `PROMISED_TOTAL_QUANTITY` property of a `LINE_ITEM` document (where is this document), represented by the variable ` myDoc ', equals ` 100 '. Otherwise, it returns false.

Example 2:

```
<IS_TRUE Value="{int($myVar/@Value)<int($myDoc/PROMISED_TOTAL_QTY/
@Value)}"/>
```

The above statement evaluates to true if the value of ` myVar ' is less than the value of `PROMISED_TOTAL_QUANTITY` property of a `LINE_ITEM` document, represented by the variable ` myDoc '. Otherwise, it returns false.

Example 3:

```
<IS_TRUE Value="{int($myDoc/PROMISED_TOTAL_QTY/@Value)<=int($myDoc/
REQUESTED_TOTAL_QTY/@Value)}"/>
```

The above statement will evaluate to true if the value of `PROMISED_TOTAL_QUANTITY` property of a `LINE_ITEM` document, represented by the variable ` myDoc ' is less than or equal to the value of `REQUESTED_TOTAL_QUANTITY` property of the same `LINE_ITEM` document. Otherwise, it returns false.

## Compound Condition

A compound condition can be formed by logically grouping AND/OR condition statements.

For example:

```
<OR>

<IS_TRUE Value="{$lineItemDoc/STATUS/@Value='Ack'}"/>

<AND>

<IS_TRUE Value="{$lineItemDoc/STATUS/@Value='Open'}"/>

<IS_TRUE Value="{$lineItemDoc/PROMISED_TOTAL_QTY/
@Value=$lineItemDoc/REQUESTED_TOTAL_QTY/@Value}"/>

</AND>

</OR>
```

The above condition is equivalent to:

```
(lineItemDoc.STATUS == Ack)

OR

(

(lineItemDoc.STATUS == Open)

AND
```

```
(lineItemDoc.PROMISED_TOTAL_QTY == lineItemDoc.REQUESTED_TOTAL_QTY)
)
```

# Business Requests

Stated simply, a Business Request consists of one or more X-Rules, each containing a set of conditions and actions. The role of a Business Request is similar that of a function in a programming language. It is called by an external entity, may take a parameter as input (usually a business document), perform certain actions, and return an output.

Note that it is not mandatory for a Business Request to take any input parameters.

There are six different types of Requests that can be defined in the MDM framework:

## DEFINE_METHOD

This is the most generic form of a business request and contains one or more X-Rules. It is invoked through the `REQUEST` tag.

## DEFINE_NOTIFICATION

The `DEFINE_NOTIFICATION` request is used to perform a set of operations on multiple documents that match the specified criteria. It contains one or more X-Rules and is invoked using the `SEND_NOTIFICATION` tag.

## DEFINE_PRE_CREATE

This is a special type of request which can be defined only once for each X-Document. `DEFINE_PRE_CREATE` is invoked just before a new instance of the X-Document is added to the Database. It does not take any parameters and does not return anything.

## DEFINE_POST_CREATE

This is a special type of request which can be defined only once for each X-Document. `DEFINE_POST_CREATE` is invoked just after a new instance of the X-Document is added to the Database. It does not take any parameters and does not return anything.

## DEFINE_PRE_MODIFY

This is a special type of request which can be defined only once for each X-Document. `DEFINE_PRE_MODIFY` is invoked just before a `MODIFY_DOCUMENT` command on an X-Document, is executed on the Database. It does not take any parameters and does not return anything.

## DEFINE_POST_MODIFY

This is a special type of request which can be defined only once for each X-Document. `DEFINE_POST_MODIFY` is invoked just after a `MODIFY_DOCUMENT` command on an X-

Document, is executed on the Database. It does not take any parameters and does not return anything.

## DEFINE_INIT

This is a special type of request which can be defined only once for each X-Service. `DEFINE_INIT` is used by the system to perform initialization tasks when the X-Service is deployed.

### Configuring custom DEFINE_INIT Rules

Perform the following steps to configure custom DEFINE_INIT rules:

1 Invoke the user defined xrule from “customInit” xrule available in customRules.xml at <MDM_Install_Directory>\cfg\xservice\bpemeta\rules.

This customInit xrule is a placeholder for all custom defined init rules and is being invoked from DEFINE_INIT rule of BPE_META service..

## DEFINE_LISTENER

`DEFINE_LISTENER` is used to setup the listeners for business events. When the event occurs, the actions specified in the `<ACTIONS>` block are triggered.

Detailed description of each of these business requests are given in the later sections of this chapter. Each section contains four sub-sections:

- Description: The usage of the request.
- Syntax: Grammar used to define the request.
- Attributes: Elements of the request definition.
- Example(s): Illustrate how the request can be used.

### Access Levels

*Description*:

For each business request, you can set the access level to specify which application and/or X-Service can access it. There are 3 possible levels:

- Public: Can be invoked by all clients.
- Protected: Can only be invoked by the owning Xservice and other co-located X-Services (default access level.)
- Private: Can be invoked only by the owning Xservice.

You can enforce the access levels by specifying the SecurityCheck attribute as 'true' in the X-Service configuration file. By default this is set to "false".

Example 1:

<service-config Name="USER_SECURITY" TransactionCache="true" SecurityCheck="true" >

.....

</service-config>

### *DEFINE_METHOD*

*Description*:

This is the most common usage of X-Rules and is representative of the generic request/response mechanism. Usually, each `DEFINE_METHOD` contains one X-Rule block. However, multiple X-Rules can be defined within one DEFINE_METHOD when there is a need to have separate sets of conditions and associated actions. The request is invoked using the `REQUEST` tag.

The following pre-defined variables can be referenced in a `DEFINE_METHOD`:

- thisParam: This variable references the parameters that are passed to the business request through the `REQUEST` tag (is of type document variable or DocVar).
- thisReturn: This variable is used to set the properties and attributes of the output from the `DEFINE_METHOD` implementation (also of type DocVar).
- thisContext: This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system.

Scope of variables: All variables (including the pre-defined ones) have the scope from the point at which they are defined to the end of the `DEFINE_METHOD` block.

*Syntax:*

```
<DEFINE_METHOD Name="methodName">
<RULE>
<COND>
{Condition statement1}
.............
{Condition statementN}
</COND>
<ACTION>
{action statement1}
.............
{action statementN}
</ACTION>
</RULE>
<RULE>
.............
.............
```

```
</RULE>

.............

</DEFINE_METHOD>
```

*Attributes:*

- Name (Required): Method name.
- Access (Optional): Sets the access level of the request to one of three possible values:
- Public: Can be invoked by all clients.
- Protected: Can only be invoked by the owning X-Service and other co-located Xservices. This is the default access level.
- Private: Can be invoked only by the owning X-Service.
- Category (Optional): Used to classify the business request.

*Example:*

The DEFINE_METHOD can be implemented as follows:

```
<DEFINE_METHOD Name="isEnterpriseRegistered">

<RULE>

<COND>

<IS_TRUE Value="{$thisParam/ENTERPRISE_ID/@Value=null}"/>

</COND>

<ACTION>

<EXCEPTION Value="MISSING_MANDATORY_INPUT">

<ARG0 Value="ID"/>

</EXCEPTION>

</ACTION>

</RULE>

<RULE>

<ACTION>

<!-- query the company table for this enterprise -->

<GET_DOCUMENT Name="COMPANY" AssignToVar="myList">

<ID Value="{$thisParam/ENTERPRISE_ID/@Value}"/>

</GET_DOCUMENT>

<!-- See if the query fetched atleast one record -->

<IF_TEST Test="count($myList/*)}>0"/>

<THEN>
```

```
<SET DocVar="$thisReturn" Property="IS_REGISTERED"
FromValue="true"/>
<SET DocVar="$thisReturn" Property="ENTERPRISE_ID"
FromValue="{$thisParam/ENTERPRISE_ID/@Value}"/>
</THEN>
<ELSE>
<SET DocVar="$thisReturn" Property="IS_REGISTERED"
FromValue="false"/>
</ELSE>
</IF_TEST>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

### *DEFINE_NOTIFICATION*

*Description:*

DEFINE_NOTIFICATION is a business request that is used to perform a set of operations on documents that match specified criteria. It contains one or more X-Rules and is invoked on the same or a different X-Service with the SEND_NOTIFICATION tag.

The following pre-defined variables can be referenced in DEFINE_NOTIFICATION:

- thisDoc: This variable references the document for which the notification is requested.
- thisParam: This variable references the parameters that are passed to the business request through the SEND_NOTIFICATION tag (is of type document variable or DocVar).
- thisReturn: This variable is used to set the properties and attributes of the output from the DEFINE_NOTIFICATION implementation (is of type DocVar).
- thisContext: This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system.

Scope of variables: All variables (including the pre-defined ones) have the scope from the point at which they are defined to the end of the DEFINE_NOTIFICATION block.

**Note:** To use this tag, ensure that in your X-Service file, the <docDefnFiles> block occurs before the <ruleDefnFiles> block. For e.g.:

```
<dom-config>
......
< docDefnFiles>
......
```

```
</ docDefnFiles>
<ruleDefnFiles>
......
</ruleDefnFiles>
......
</dom-config>
```

*Syntax:*

```
<DEFINE_NOTIFICATION Name="noticeName" Document="docName">
<RULE>
<COND>
{Condition statement1}
.............
{Condition statementN}
</COND>
<ACTION>
{action statement1}
.............
{action statementN}
</ACTION>
</RULE>
<RULE>
.............
.............
</RULE>
</DEFINE_NOTIFICATION>
```

*Attributes:*

- Name (Required): Notification name.
- Document (Required): X-Document name for which notification is called.
- Comment (Optional): A brief description of the notification.
- Category (Optional): To classify the request.

*Example:*

The DEFINE_NOTIFICATION can be implemented as follows:

```
<DEFINE_NOTIFICATION Name="creditDeniedActionHandler"
Document="CUSTOMER_ORDER">
<RULE>
<ACTION>
<!-- Build the message -->
<TO_DOCVAR AssignToVar="messageData">
<MESSAGE_DATA>
<TO_XML DocVar="thisDoc"/>
</MESSAGE_DATA>
</TO_DOCVAR>
<!-- Send email -->
<REQUEST Name="sendEmails">
<SUBJECT Value="Credit denied"/>
<MESSAGE>
<XML_DATA XsltFileName="oa.xsl">
<CREDIT_DENIED>
<TO_XML DocVar="messageData"/>
</CREDIT_DENIED>
</XML_DATA>
</MESSAGE>
<TO_XML SelectList="$thisParam/LISTENERS"/>
</REQUEST>
<!-- Send alert -->
<REQUEST Name="sendAlerts">
<ALERT_PROPERTIES>
<TYPE_ID Value="CREDIT_DENIED_CO_ALERT"/>
<CO_ID Value="{$thisDoc/ID/@Value}"/>
<ALERT_TEXT Value="Credit denied; Order rejected."/>
</ALERT_PROPERTIES>
<TO_XML SelectList="$thisParam/LISTENERS"/>
</REQUEST>
<!-- Send to integration -->
<REQUEST Name="sendEDIMessage">
```

```
<AP_ID Select="$thisParam/AP_ID/@Value"/>

<TO_XML DocVar="messageData"/>

<TO_XML SelectList="$thisParam/LISTENERS"/>

</REQUEST>

</ACTION>

</RULE>

</DEFINE_NOTIFICATION>
```

### *DEFINE_PRE_CREATE*

*Description:*

This is a special X-Rule which can be defined ONLY ONCE for an X-Document. It is invoked just before a new instance of the X-Document is added to the Database.

`DEFINE_PRE_CREATE` is useful for performing actions like automatically generating an id, setting the creation date of document or setting some property values before adding the document to the database. It does not take any parameters and does not return anything. This rule is automatically called by the system using the OnPreCreate handler, and does not accept an explicit invocation by a user.

The following pre-defined variables can be referenced in `DEFINE_PRE_CREATE`:

- thisDoc. This variable references the document for which the `DEFINE_PRE_CREATE` is called (is of type DocVar).
- thisContext. This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system. It can be used as a parameter to a Native function.
- thisParentDoc. This variable is available only if the document is being added as a nested document, and points to the enclosing (or parent) document (is of type DocVar).

Scope of Variables: All variables, including the pre-defined ones, have the scope from the point at which they are defined to the end of the `DEFINE_PRE_CREATE` block.

**Note:** To use this tag, ensure that in your X-Service file, the `<docDefnFiles>` block occurs before the `<ruleDefnFiles>` block. For example:

```
<dom-config>

......

< docDefnFiles>

......

</ docDefnFiles>

<ruleDefnFiles>

......

</ruleDefnFiles>
```

```
......
</dom-config>
```

*Syntax:*

```
<DEFINE_PRE_CREATE Document="docName">
<RULE>
<COND>
{ Condition statement1}
.............
{ Condition statementN}
</COND>
<ACTION>
{ action statement1}
.............
{ action statementN}
</ACTION>
</RULE>
<RULE>
<COND>
.............
</COND>
<ACTION>
.............
</ACTION>
</RULE>
............
</DEFINE_PRE_CREATE>
```

*Attributes:*

Document (Required): X-Document name

*Example:*

The following pre-create rule is used to check if the mandatory fields are populated in the document being added to the `LINE_ITEM` table, as well as to automatically generate an id for the field used for the primary key. `LINE_ITEM` is a nested document, whose parent is the `PURCHASE_ORDER` document.

```
<DEFINE_PRE_CREATE Document="LINE_ITEM">
<RULE>
<!--Check to see if LINE_ITEM has all the mandatory property values-
->
<COND>
<OR>
<IS_TRUE Value="{$thisDoc/SELLER_ID/@Value=null}"/>
<IS_TRUE Value="{$thisDoc/ITEM_ID/@Value=null}"/>
<IS_TRUE Value="{$thisDoc/PO_ID/@Value=null}"/>
<IS_TRUE Value="{$thisDoc/BUYER_ID/@Value=null}"/>
</OR>
</COND>
<ACTION>
<EXCEPTION Value="MISSING_MANDATORY_FIELDS"/>
</ACTION>
</RULE>
<RULE>
<!--If LINE_ITEM does not have pre-defined id, then auto-generate the
ID-->
<COND>
<IS_TRUE Value="{$thisDoc/ID/@Value=null}"/>
</COND>
<ACTION>
<!--autogenerate the id -->
<CREATE_DOCUMENT_ID Name="LINE_ITEM" AssignToVar="uid"/>
<SET DocVar="thisDoc" Property="ID" FromVar="uid"/>
</ACTION>
</RULE>
</DEFINE_PRE_CREATE>
```

### *DEFINE_POST_CREATE*

*Description:*

This is a special X-Rule which can be defined ONLY ONCE for an X-Document. It is invoked just after a new instance of the X-Document is added to the Database. `DEFINE_POST_CREATE` is useful for performing actions like sending out an alert, setting up a

timer, create another document, etc. It does not take any parameters and does not return anything. This rule is automatically called by the system using the OnPostCreate handler, and does not accept an explicit invocation by a user.

The following pre-defined variables can be referenced in `DEFINE_POST_CREATE`:

- thisDoc. This variable references the document for which the `DEFINE_POST_CREATE` is called (is of type DocVar).
- thisContext. This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system. It can be used as a parameter to a Native function.
- thisParentDoc. This variable is available only if the document is being added as a nested document, and points to the enclosing (or parent) document (is of type DocVar).

Scope of Variables: All variables, including the pre-defined ones, have the scope from the point at which they are defined to the end of the `DEFINE_POST_CREATE` block.

**Note:** To use this tag, ensure that in your X-Service file, the <docDefnFiles> block occurs before the <ruleDefnFiles> block. For example:

```
<dom-config>
......
< docDefnFiles>
......
</ docDefnFiles>
<ruleDefnFiles>
......
</ruleDefnFiles>
......
</dom-config>
```

*Syntax:*

```
<DEFINE_POST_CREATE Document="docName">
<RULE>
<COND>
{ Condition statement1}
..............
{ Condition statementN}
</COND>
<ACTION>
{ action statement1}
```

```
.............
{ action statementN}
</ACTION>
</RULE>
<RULE>
<COND>
.............
</COND>
<ACTION>
.............
</ACTION>
</RULE>
............
</DEFINE_PRE_CREATE>
```

*Attributes:*

Document (Required): X-Document name

*Example:*

The following post-create rule is used for generating an alert whenever a new SALES_ORDER document is added to the database.

```
<DEFINE_POST_CREATE Document="SALES_ORDER">
<RULE>
<ACTION>
<!--Write an ALERT to the Seller-->
<ADD_DOCUMENT Name="ALERT">
<TO_USER Value="{$thisDoc/SELLER_ID/@Value}"/>
<FROM_USER Value="{$thisDoc/BUYER_ID/@Value}"/>
<ALRT_LEVEL Value="SALES_ORDER"/>
<SO_ID Value="{$thisDoc/ID/@Value}"/>
<MSG Value="New Sales Order created by the system"/>
</ADD_DOCUMENT>
</ACTION>
</RULE>
</DEFINE_POST_CREATE>
```

### *DEFINE_PRE_MODIFY*

*Description:*

This is a special X-Rule which can be defined ONLY ONCE for an X-Document. It is invoked just before a `MODIFY_DOCUMENT` command on an X-Document, is executed on the Database. `DEFINE_PRE_MODIFY` is useful for performing actions like automatically updating last-modified-date, last-modified-by-user or setting some property values before updating the document on the database. It does not take any parameters and does not return anything. This rule is automatically called by the system using the OnPersist handler, and does not accept an explicit invocation by a user.

The following pre-defined variables can be referenced in `DEFINE_PRE_MODIFY`:

- thisDoc. This variable references the modify document xml for which the `DEFINE_PRE_MODIFY` is called (is of type DocVar).
- thisContext. This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system. It can be used as a parameter to a Native function.

Scope of Variables: All variables, including the pre-defined ones, have the scope from the point at which they are defined to the end of the `DEFINE_PRE_MODIFY` block.

*Syntax:*

```
<DEFINE_PRE_MODIFY Document="docName">
<RULE>
<COND>
{ Condition statement1}
.............
{ Condition statementN}
</COND>
<ACTION>
{ action statement1}
.............
{ action statementN}
</ACTION>
</RULE>


............
</DEFINE_PRE_MODIFY>
```

*Attributes:*

Document (Required): X-Document name

*Example:*

The following pre-modify rule is used for updating LAST_MODIFIED_DATE and LAST_MODIFIED_BY_ID columns whenever a MODIFY_DOCUMENT command is issued on RECEIPT_LINE x-doc.

```
<DEFINE_PRE_MODIFY Document="RECEIPT_LINE">
   <RULE>
       <ACTION>
          <SET Var="updateProps" FromValue="{$thisDoc/
UPDATE_PROPERTIES}"/>
             <SET DocVar="updateProps" Property="LAST_MODIFIED_DATE"
FromValue="{date()}"/>
               <SET DocVar="updateProps"
Property="LAST_MODIFIED_BY_ID" FromValue="{$thisUser}"/>
          </ACTION>
      </RULE>
   </DEFINE_PRE_MODIFY>
```

### *DEFINE_POST_MODIFY*

*Description:*

This is a special X-Rule which can be defined ONLY ONCE for an X-Document. It is invoked just after a MODIFY_DOCUMENT command on an X-Document, is executed on the Database. It does not take any parameters and does not return anything. This rule is automatically called by the system using the OnPersist handler, and does not accept an explicit invocation by a user.

The syntax and the pre-defined variables available are identical to the DEFINE_PRE_MODIFY statement. The only difference is the tag name.

### *DEFINE_INIT*

*Description:*

This is a special X-Rule which can be defined ONLY ONCE for each X-Service and is used by the system to perform initialization tasks when the X-Service is deployed. It does not take any parameters and does not return anything. This rule is automatically called by the system using the OnPostCreate handler, and does not accept an explicit invocation by a user.

The following pre-defined variables can be referenced in DEFINE_INIT:

thisContext. This variable is an instance of the Java class com.teradata.X-Core.tool.OmxContext. It provides methods to access the database connections and documents in the system. It can be used as a parameter to a Native function.

Scope of Variables: All variables, including the pre-defined ones, have the scope from the point at which they are defined to the end of the DEFINE_POST_CREATE block.

*Syntax:*

```
<DEFINE_INIT>
<RULE>
<COND>
{ Condition statement1}
.................
{ Condition statementN}
</COND>
<ACTION>
{ action statement1}
.................
{ action statementN}
</ACTION>
</RULE>
<RULE>
...................
...................
</RULE>
.............
</DEFINE_INIT>
```

*Attributes:*

None

*Example:*

The following init rule prints a status message to the log whenever the X-Service is started.

```
<DEFINE_INIT>
<RULE>
<ACTION>
<PRINTLN Value="Starting Service"/>
</ACTION>
</RULE>
</DEFINE_INIT>
```

### *DEFINE_LISTENER*

*Description:*

A Business Request expressed using `DEFINE_LISTENER` is invoked when a certain pre-defined business event occurs and all the associated X-Rules are executed. It only takes one parameter as input - the name of the event descriptor.This rule is registered with the MDM server during startup and is automatically called by the system when the event occurs.

*Syntax:*

```
<DEFINE_LISTENER Name="Name" Descriptor="//ServiceName/Descriptor">
<RULE>
<COND>
{ Condition statement1}
.................
{ Condition statementN}
</COND>
<ACTION>
{ action statement1}
.................
{ action statementN}
</ACTION>
</RULE>
<RULE>
...................
...................
</RULE>
............
</DEFINE_INIT>
```

*Attributes:*

- Name (Required): Business Request name
- Descriptor (Required): Event descriptor name

*Example:*

The following listener rule prints a status message to the log whenever the event occurs.

```
<DEFINE_LISTENER Name="newOrderListener"
Descriptor="newOrderCreated">
<RULE>
```

```
<ACTION>
<PRINTLN Value="New order created with id: {$thisParam/bookId/
@Value}"/>
</ACTION>
</RULE>
</DEFINE_INIT>
```

# Invoking X-Rules

For each of the business requests that can be invoked by a user, there is a specific mechanism for calling it:

- REQUEST: used with `DEFINE_METHOD`
- SEND_NOTIFICATION: used with `DEFINE_NOTIFICATION`

## REQUEST

*Description:*

The `REQUEST` statement is used to call the business request defined using `DEFINE_METHOD` tags. The request can be executed synchronously or asynchronously. The asynchronous processing should be used only if the return value of the request is not important.

*Syntax:*

```
<REQUEST Name="methodName" {Additional Attributes - optional}>
{XML tags - optional}
</REQUEST>
```

*Attributes:*

- Name (Required): Name of method being called.
- ServiceName (Optional): Service in which method is defined. If not specified, the owner X-Service is taken as default.
- AssignToVar (Optional): Variable to assign the response of the request. If not defined then the response is ignored.
- Synchronous (Optional): yes/no. If not specified, 'yes' is taken as default. To define synchronous and asynchronous workflows.
- MaxRows (Optional): The maximum number of records that should be returned by the request.
- ReturnRowCount (Optional): The count of records returned by the request.
- HandleException (Optional): true/false.

*Example:*

Sample request:

```
<REQUEST Name="isEnterpriseRegistered">
<ENTERPRISE_ID Value="COMP_9876589"/>
</REQUEST>
Sample response:
<RESPONSE Name="isEnterpriseRegistered">
<IS_REGISTERED Value="true"/>
<ENTERPRISE_ID Value="COMP_9876589"/>
</RESPONSE >
```

Note that the variables like thisParam can be used within the rule to access the `<XML TAGS>` passed using a Request call.

## SEND_NOTIFICATION

*Description:*

The `SEND_NOTIFICATION` statement is used to call a document notification, which may be defined within the same X-Service or on a different service. The request can be executed synchronously or asynchronously (using the Synchronous attribute). However, asynchronous processing should be used only if the return value of the request is not important. The following tasks are performed by the system when it encounters a `SEND_NOTIFICATION` request:

- All the documents that match the criteria specified in the `DOCUMENT_CONTEXT` of the `SEND_NOTIFICATION` are selected.
- For each selected document, the property values specified in `UPDATE_PROPERTIES` are applied.
- `DEFINE_NOTIFICATION` is then invoked for each document. The document for which the `DEFINE_NOTIFICATION` is invoked can be accessed through thisDoc variable in the rules.

*Syntax:*

```
<SEND_NOTIFICATION Name="noticeName" HandleException ="yes/no">
<FOR_DOCUMENT Name="documentName">
<DOCUMENT_CONTEXT>
{query criteria formed using lhsExpression}
</DOCUMENT_CONTEXT>
<UPDATE_PROPERTIES>
<PropertyName1 lhsExpression1>
.....................................................
<PropertyNameN lhsExpressionN>
```

```
</UPDATE_PROPERTIES>

</FOR_DOCUMENT >

</SEND_NOTIFICATION >
```

*Attributes:*

- `Name` (Required): Notification Name.
- `ServiceName` (Optional): Name of the service where Notification is defined. If not specified, the owner X-Service is taken as default.
- `Synchronous` (Optional): yes/no.
- `AssignToVar` (Optional): Variable to which the response is assigned. If not defined then the response is ignored.
- `HandleException` (Optional): true/false

*Example:*

```
<SEND_NOTIFICATION Name="SendInvoiceInfo" AssignToVar="rtnVar">

<FOR_DOCUMENT Name="SHIPMENT">

<DOCUMENT_CONTEXT>

<ID Value="{$shipmentDoc/ID/@Value}"/>

</DOCUMENT_CONTEXT>

<UPDATE_PROPERTIES>

<PAYER_ID Value="{$profileDoc/ID/@Value}"/>

<PAYEE_ID Value="giri"/>

<SH_CHARGES Value="{$shCharges}"/>

<OTHER_CHARGES Value="40"/>

<INVOICE_VALUE Value="{$invValue}"/>

<TAX Value="10"/>

<PAY_METHOD Value="Credit"/>

</UPDATE_PROPERTIES>

</FOR_DOCUMENT>

</SEND_NOTIFICATION>
```

# CHAPTER 9 X-Operations

### What's In This Chapter

This chapter discusses in detail the X-Operations available in the MDM framework.

Topics include:


- Introduction
- X-Document Operations
- Smart XCommands
- SQL Operations
- XML Manipulation Operations
- Logical Operations
- User Interface Related Operations
- Utility Operations
- Framework Service Operations


## Introduction

X-Operations are action statements that are used to write the business logic inside a workflow in MDM. As you will recall, the flow logic of a workflow is represented using nodes such as Start, Task, Event, Branch, UI, Done etc. Within each node, X-Operations are used to execute specific tasks. The logic of a business rule (or X-Rule) is also written using the same X-Operations. Based on the type of action performed, X-Operations are classified into:

- X-Document
- SQL
- XML Manipulation
- Logic
- User-Interface
- Utility
- Framework Services

# X-Document Operations

## ADD_DOCUMENT

*Description:*

The `ADD_DOCUMENT` operation allows creation of a new document in the system. The document supplied in the request can be a nested structure. The nested document should be related to the enclosing document through document link. The response contains the primary key values (document identifiers) of the top-level document.

Documents can be nested to any level depth. The enclosed document should be related to the enclosing through a document link. The enclosed document derives the values of mapped property values from the enclosing document by the system.

`ADD_DOCUMENT` can result in MDM caching the newly added document in memory and only persisting it when the transaction ends. To avoid caching and to persist the document as soon as the add document operation is executed, users can specified an attribute on `ADD_DOCUMENT` called `CacheHint="2"`. When this hint is specified, MDM will make a database insert upon the `ADD_DOCUMENT` call.

*Syntax*:

```
<ADD_DOCUMENT Name="XdocName">
```

*Attributes:*

- Name (Required): Name of the X-Document that you want to add.
- ServiceName (Optional): Service in which the X-Document is defined. If not specified, the owner X-Service is taken as default.
- AssignToVar (Optional): Variable to which the response is assigned.
- CacheHint (Optional): A value of '2' guarantees that the document is inserted into the database immediately.

*Example 1:*

```
<REQUESTS>
   <ADD_DOCUMENT Name="PURCHASE_ORDER">
      <ID Value="PO_123"/>
      <BUYER_ID Value="buyer01"/>
      <CREATION_DATE Value="04/20/2000"/>
      <PROMISED_DELIVERY_TIME Value="09/17/2000 15:30:42"/>
      <STATE Value="open"/>
      <PIN Value="1"/>
      <INVOICE Value="7000.0"/>
      <MY Value="7000.0"/>
```

```
      <MY_ATTRIBUTE_A Value="This is a user attribute"/>
      <MY_ATTRIBUTE_B Value="Another user attribute"/>
   </ADD_DOCUMENT>
</REQUESTS>
```

The response would be as follows:

For successful execution of request:

```
<RESPONSES>
   <RESPONSE Status="success" Description="Add Document Successful">
      <ID Value="PO_123"/>
   </RESPONSE>
</RESPONSES>
```

For unsuccessful execution of request:

```
<RESPONSES Status="error">
   <RESPONSE Status="error" Description="ORA-1011: Primary Key
Constraint Violated">
      <ID Value="PO_123"/>
   </RESPONSE>
</RESPONSES>
```

*Example 2:*

This is a modified example of the previous example with `LINE_ITEM` instances.

```
<REQUESTS>
   <ADD_DOCUMENT Name="PURCHASE_ORDER"/>
      <BUYER_ID Value="buyer01"/>
      <PROMISED_DELIVERY_TIME Value="09/17/2000 15:30:42"/>
      <STATE Value="open"/>
      <PIN Value="1"/>
      <INVOICE Value="7000.0"/>
      <MY Value="7000.0"/>
      <MY_ATTRIBUTE_A Value="This is a user attribute"/>
      <MY_ATTRIBUTE_B Value="Another user attribute"/>
      <LINE_ITEMS>
      <LINE_ITEM>
         <ITEM_ID Value="item1-1-2"/>
```

```
            <SELLER_ID Value="seller01"/>
            <PROMISED_TOTAL_QTY Value="50"/>
            <STATE Value="Open"/>
         </LINE_ITEM>
         <LINE_ITEM>
            <ITEM_ID Value="item9-6-2"/>
            <SELLER_ID Value="seller02"/>
            <PROMISED_TOTAL_QTY Value="200"/>
            <STATE Value="Open"/>
         </LINE_ITEM>
           </LINE_ITEMS>
         </ADD_DOCUMENT>
</REQUESTS>
```

The response would be as follows:

```
</RESPONSES>
  <RESPONSE Status="success" Description="Add Document Successful">
      <ID Value="PO-7854347890"/>
   </RESPONSE>
</RESPONSES>
```

In the above example, the `LINE_ITEM` is related to the `PURCHASE_ORDER` through `LINE_ITEMS` document link defined on the `PURCHASE_ORDER`. The `LINE_ITEM` instances derive the value of their `PO_ID` property (mapped property) from the enclosing `PURCHASE_ORDER` document (ID property).

## CREATE_DOCUMENT_ID

*Description:*

The `CREATE_DOCUMENT_ID` operation automatically creates an ID for the specified document; very useful for generating surrogate keys. It is used in conjunction with a pre-defined ID generation scheme and the 'idgen' tags; see the chapter on ID Generation for more details.

*Syntax:*

```
<CREATE_DOCUMENT_ID Name="docName" AssignToVar="varName"/>
```

*Attributes:*

- Name (Required): Document for which ID is generated.
- AssignToVar (Required): Variable to which the generated ID is assigned.

- HandleException (Optional):'true/false'; to specify if MDM should catch exceptions.

*Example:*

```
<CREATE_DOCUMENT_ID Name="CUSTOMER_ORDER" AssignToVar="newID"/>
```

## QUERY CRITERIA

A query criterion is the filter specification that is used to identify a specific set of documents/ records. You can use it while fetching records from database table. A simple criterion has the following format:

```
<{PROPERTY_NAME} Value="{value}" MatchBy="{Match By Operator}">
```

This criterion compares the value of the Property against the value provided, using the MatchBy operator. The following is a list of valid MatchBy operators with examples.

- Starts with (for Partial match)

  ```
  <ITEM_NAME Value ="Muff" MatchBy="STARTS_WITH"/>
  ```

- Ends With (for Partial Match)

  ```
  <ITEM_NAME Value ="ffler" MatchBy="ENDS_WITH"/>
  ```

- Contains (for Partial Match)

  ```
  <ITEM_NAME Value ="ffl" MatchBy="CONTAINS"/>
  ```

- Less Than Or Equal To (<=)

  ```
  <TOTAL_QTY Value ="50" MatchBy="LESS_EQUAL"/>
  ```

- Greater Than or Equal To (>=)

  ```
  <TOTAL_QTY Value ="50" MatchBy="GREATER_EQUAL"/>
  ```

- Less Than (<)

  ```
  <CREATION_DATE Value="05/30/2000" MatchBy="LESS"/>
  ```

- Greater Than (>)

  ```
  <CREATION_DATE Value="05/30/2000" MatchBy="GREATER"/>
  ```

- Exact Match (=)

  ```
  <ITEM_ID Value ="Muffler" MatchBy="EQUAL"/>
  ```

- Not Equals (!=)

  ```
  <ITEM_ID Value ="Muffler" MatchBy="NOT_EQUAL"/>
  ```

You can compare one property against the other using the following format:

```
<{PROPERTY_NAME} Property="{another property name}" MatchBy="{Match
By Operator}">
```

For example:

```
<PROMISED_TOTAL_QTY Property="DELIVERED_TOTAL_QTY"
MatchBy="GREATER">
```

represents:

```
PROMISED_TOTAL_QTY > DELIVERED_TOTAL_QTY
```

The logical AND/OR operators can be used to combine simple query criteria to create compound query criteria.

For example:

```
<AND>
<ITEM_ID Value="Muffler" MatchBy="EQUAL"/>
<OR>
      <SUPPLIER_ID Value="VIP DRAM" MatchBy="EQUAL"/>
      <SUPPLIER_ID Value="Synekdo" MatchBy="EQUAL"/>
</OR>
</AND>
```

The compound expression represents the criteria:

```
ITEM_ID == "Muffler" AND (SUPPLIER_ID == "VIP DRAM" OR SUPPLIER_ID ==
"Synekdo")
```

## REMOVE_DOCUMENT

*Description:*

The `REMOVE_DOCUMENT` operation allows one or more documents that match query criteria to be deleted. All the linked document instances defined with the AutoDelete flag turned on are also deleted. This operation is performed asynchronously.

The 'RowCount' attribute in the response XML indicates the number of deleted instances of document.

*Syntax:*

```
<REMOVE_DOCUMENT Name="doc-name">
{A simple or compound query criteria}
</REMOVE_DOCUMENT>
```

*Attributes:*

Name (Required): Document to be deleted.

*Example 1:*

```
<REMOVE_DOCUMENT Name="PURCHASE_ORDER">
<BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
</REMOVE_DOCUMENT>
```

The response for a successful execution:

```
<RESPONSE Status="success" RowCount="3" Description="Remove Document
Successful"/>
```

The response for an un-successful execution:

```
<RESPONSE Status="error" Description="Document: PURCHASE_ORDER does
not exist">
```

## MODIFY_DOCUMENT

*Description:*

This operation allows one or more documents that match the query criteria to be modified.

All documents matching the query criteria are modified with the values of the PropertyName tags specified in `UPDATE_PROPERTIES`. The PropertyName tags in `UPDATE_PROPERTIES` should belong to the document specified. There should be atleast one PropertyName tag under `UPDATE_PROPERTIES`. If nothing is present, an exception may be raised.

The RowCount attributes returned in a successful response XML indicates the number of modified instances of the document.

### Bypassing MDM Cache

`MODIFY_DOCUMENT` can result in MDM caching the modified document in memory and persisting it only when the transaction ends. To avoid caching and to persist the document as soon as the modify document operation is executed, users can specified an attribute on `MODIFY_DOCUMENT` called `CacheHint="2"`. When this hint is specified, MDM will make a database commit upon the modify_document call.

*Syntax:*

```
<MODIFY_DOCUMENT Name="documentName">

<DOCUMENT_CONTEXT>

{A Simple or Compound query criteria}

</DOCUMENT_CONTEXT>

<UPDATE_PROPERTIES>

<PropertyName1 Value="value1"/>

.............

</UPDATE_PROPERTIES>

</MODIFY_DOCUMENT>
```

*Attributes:*

- Name (Required): Document to be modified.
- CacheHint (Optional): When set to '2', MDM will perform a database commit at the end of the MODIFY_DOCUMENT operation.

*Example 1:*

The following request changes the state of `PURCHASE_ORDER` instances of buyer01 to "hold":

```
<MODIFY_DOCUMENT Name="PURCHASE_ORDER">

<DOCUMENT_CONTEXT>
```

```
<BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
</DOCUMENT_CONTEXT>
<UPDATE_PROPERTIES>
<STATE Value="hold"/>
</UPDATE_PROPERTIES>
</MODIFY_DOCUMENT>
```

The response for a successful execution:

```
<RESPONSE Status="success" RowCount="3" Description="Modify Document
Successful"/>
```

The response for an unsuccessful execution:

```
<RESPONSE Status="error" Description="Document: PURCHASE_ORDER does
not exist">
```

*Example 2:*

You can also use expressions in the document context and update properties section of modify document.

See section on Expression in `GET_DOCUMENT` before reading this example.

```
<MODIFY_DOCUMENT Name="ORG_RELN">
      <DOCUMENT_CONTEXT>
        <RELN_VAL Expression="#|RELN_VAL|#*2 - 10"/>
      </DOCUMENT_CONTEXT>
      <UPDATE_PROPERTIES>
        <RELN_VAL Expression="#|RELN_VAL|#*2">
      </UPDATE_PROPERTIES>
</MODIFY_DOCUMENT>
```

## SAVE

*Description:*

The `SAVE` operation can be used to save an existing document instance to the database.

*Syntax:*

```
<SAVE DocVar="myVar" ServiceName="service" Synchronous="yes/no"/>
```

*Attributes:*

- DocVar (Required): Variable referencing the XML document that is saved to the database.
- ServiceName (Optional): Service that owns the document instance; if not specified, then the current service is taken as default.

- Synchronous (Optional): yes/no. If 'yes', then the save is performed right away, otherwise it is done asynchronously. The default value is 'yes'.

*Example 1:*

```
<!--CREATE a new PO instance-->
<TO_DOCVAR AssignToVar="myPODoc">
<PURCHASE_ORDER>
<ID Value="{$thisParam/ID/@Value}"/>
<BUYER_IDValue="buyer01"/>
<CREATION_DATE Value="{date()}"/>
<STATUS Value="Created"/>
</PURCHASE_ORDER>
</TO_DOCVAR>
<SAVE DocVar="myPODoc"/>
```

*Example 2:*

```
<!--GET a PO instance from the database-->
<GET_DOCUMENT Name="PURCHASE_ORDER" AssignToVar="myPODocs">
<ID Value="PO-123" MatchBy="EQUAL"/>
</GET_DOCUMENT>
<SET Var="myPODoc" FromValue="{$myPODocs/PURCHASE_ORDER}"/>

<!--Modify the status of the PO instance in the rules-->
<SET DocVar="myPODoc" Property="STATUS" FromValue="Hold"/>

 <!-Now, save the modified PO instance to the database through SAVE
command. The save command would internally issue an MODIFY_DOCUMENT
request.-->
<SAVE DocVar="myPODoc"/>
```

## GET_DOCUMENT

*Description:*

The `GET_DOCUMENT` operation allows a document to be retrieved based upon query criterion.

You can perform pagination by restricting the number of document instances returned by a `GET_DOCUMENT` query; use the StartAtRow and MaxRows attributes to specify this.

*Syntax:*

```
<GET_DOCUMENT Name="documentName">
<Property1 Value="..." MatchBy="...">
<Property2 Value="..." MatchBy="...">
......
<SELECT>
<PropertyA/>
<PropertyB/>
......
</SELECT>
</GET_DOCUMENT>
```

Property1, Property2, etc. values are used as a filter so that only the documents that contain these field values are retrieved from the database; if not specified, then all the documents are retrieved from the database.

The properties in the `SELECT` tags, PropertyA, PropertyB, etc. are used to specify which columns (or fields) should be retrieved by the operation; if not specified, then all the columns are retrieved from the database.

*Attributes:*

- Name (Required): The name of the X-Document to be retrieved
- ServiceName (Optional): Service from which the document is retrieved. If not specified, then owner service is taken as default.
- AssignToVar (Optional): Variable to which the response is assigned.
- StartAtRow (Optional): Starting row number from which records should be retrieved
- MaxRows (Optional): Maximum number of records to be retrieved. If this is not provided then it is defaulted to the Default MaxRows param. Currently this is 1000 rows. You can also override the default max rows param for a service by defining the service param, `DEFAULT_MAX_ROWS_FETCH`.

  For example, the following lines in service-config file will set the default max rows for that service to 20000:

  ```
  <service-params>
        <param Name="DEFAULT_MAX_ROWS_FETCH" Value="20000"/>
  </service-params>
  ```
- ReturnRowCount - yes/no (Optional): To show number of records retrieved
- HandleException - true/false (Optional): Whether the exception should be handled
- DirtyRead - true/false (Optional), default value is false: To read the underlying table applying Access Level Lock to avoid deadlocks.

*Example 1:*

The basic `GET_DOCUMENT` operation has the following format:

```
<GET_DOCUMENT Name="LINE_ITEM">
<ITEM_ID Value="Muffler" MatchBy="EQUAL"/>
</GET_DOCUMENT>
```

If no MatchBy operator is explicitly specified, then 'EQUAL' is taken as default. Hence, the above operation can be simply written as:

```
<GET_DOCUMENT Name="LINE_ITEM">
<ITEM_ID Value="Muffler"/>
</GET_DOCUMENT>
```

A successful response has the following format:

```
<RESPONSE Status="success" Description="Get Document Successful">
<LINE_ITEM>
<ID Value="LI-1254347890"/>
<ITEM_ID Value="Muffler"/>
<SUPPLIER_ID Value="VIP DRAM"/>
<PROMISED_TOTAL_QTY Value="45"/>
<CREATION_DATE Value="09/17/2000 15:30:42"/>
<STATUS Value="open"/>
</LINE_ITEM>
<LINE_ITEM>
<ID Value="LI-1254347890"/>
<ITEM_ID Value="Muffler"/>
<SUPPLIER_ID Value="Synekdo"/>
<PROMISED_TOTAL_QTY Value="75"/>
<CREATION_DATE Value="10/12/2000 15:30:42"/>
<STATUS Value="open">
</STATUS>
</LINE_ITEM>
</RESPONSE>
```

An unsuccessful response has the following format:

```
<RESPONSES Status="error" Description="Transaction Rolled Back">
<RESPONSE Status="error" Description="LINE_ITEM document not found"/
>
</RESPONSES>
```

*Example 2:*

If more than one compound or simple query criteria is used in a GET_DOCUMENT, then the criteria is combined into a single compound criteria using AND operator.

The following is an example of a GET_DOCUMENT with two simple and one compound Query Criteria.

```
<GET_DOCUMENT Name="LINE_ITEM">
<ITEM_ID Value="Muffler" MatchBy="EQUAL"/>
<OR>
<SUPPLIER_ID Value="VIP DRAM" MatchBy="EQUAL"/>
<SUPPLIER_ID Value="Synekdo" MatchBy="EQUAL"/>
</OR>
<STATUS Value="Open" MatchBy="EQUAL"/>
</GET_DOCUMENT>
```

The following GET_DOCUMENT is equivalent to the one described above,

```
<GET_DOCUMENT Name="LINE_ITEM">
<AND>
<ITEM_ID Value="Muffler" MatchBy="EQUAL"/>
<OR>
<SUPPLIER_ID Value="VIP DRAM" MatchBy="EQUAL"/>
<SUPPLIER_ID Value="Synekdo" MatchBy="EQUAL"/>
</OR>
<STATUS Value="Open" MatchBy="EQUAL"/>
</AND>
</GET_DOCUMENT>
```

*Example 3:*

The following example shows a paginating query:

```
<GET_DOCUMENT Name="PURCHASE_ORDER" StartAtRow="0" MaxRows="100"
ReturnRowCount="yes">
<CREATION_DATE MatchBy="GREATER" Value="01/01/2001"/>
</GET_DOCUMENT>
```

The above query will return the first 100 rows matching the criteria; for example:

```
<RESPONSE TotalRowCount="160">
      <PURCHASE_ORDER>
```

```
        ...
    </PURCHASE_ORDER>
        ...
</RESPONSE>
```

If the number of rows matching the query is less than 100, then it will return the actual number of rows matching the query. If the query returns 100 rows, then you can retrieve the next 100 rows (starting from row 100) by issuing the query:

```
<GET_DOCUMENT Name="PURCHASE_ORDER" StartAtRow="100" MaxRows="100">
    <CREATION_DATE MatchBy="GREATER" Value="01/01/2001"/>
</GET_DOCUMENT>
```

## Expression

*Description:*

You can also use Expression attribute to specify a query. It is required when the sql expressions are fairly complex to be expressed in the standard `GET_DOCUMENT` syntax. This is half way between using a standard `GET_DOCUMENT` and `EXECUTE_SQL_QUERY`.

The expressions are in-lined in the generated SQL. You have to ensure that the expressions provided actually make sense for the data base on which the query is issued.

Any property tokens escaped using (`#|, |#`) in the expression will be replaced with the actual database column names. The property tokens can be optionally prefixed with the table name prefixes that will appear in the general SQL. For this purpose, you can assign a prefix to the `GET_DOCUMENT` tag and each of the `QUERY_DOCUMENT_LINK` tags appearing in the GET_DOCUMENT tag.

*Example 1:*

X-Docs for this example:

```
<DOCUMENT Name="ORG">
        <PROPERTY Name="ID" PrimaryKey="true" Type="string(32)" />
        <PROPERTY Name="NAME" Type="string(32)" />
        <PROPERTY Name="OVAL" ColumnName="ORG_VAL" Type="float" />
       <DOCUMENTLINK Name="SELL_ORG_LINK" LinkedDocument="ORG_RELN"
AutoDelete="true">
            <MAP_PROPERTY To="SELL_ORG" From="ID" />
         </DOCUMENTLINK>
    </DOCUMENT>
    <DOCUMENT Name="ORG_RELN">
```

```
        <PROPERTY Name="CUST_ORG" PrimaryKey="true" Type="string(32)"
/>
        <PROPERTY Name="SELL_ORG" PrimaryKey="true" Type="string(32)"
/>
         <PROPERTY Name="RELN_VAL" Type="float" />
         <DOCUMENTLINK Name="CUST_ORG_LINK" LinkedDocument="ORG">
             <MAP_PROPERTY From="CUST_ORG" To="ID" />
         </DOCUMENTLINK>
         <DOCUMENTLINK Name="SELL_ORG_LINK" LinkedDocument="ORG">
             <MAP_PROPERTY From="SELL_ORG" To="ID" />
         </DOCUMENTLINK>
    </DOCUMENT>


    <GET_DOCUMENT Name="ORG_RELN" Expression="#|p1.OVAL|# +
#|p2.OVAL|# = #|RELN_VAL|#">
      <QUERY_DOCUMENT_LINK Name="CUST_ORG_LINK" Prefix="p1">
        <OVAL Expression="2*#|OVAL|# - 10"/>
       </QUERY_DOCUMENT_LINK>
      <QUERY_DOCUMENT_LINK Name="SELL_ORG_LINK" Prefix="p2">
        <OVAL Expression="2*#|OVAL|# - 10"/>
       </QUERY_DOCUMENT_LINK>
    </GET_DOCUMENT>
```

## ORDER_BY

*Description:*

When using the `GET_DOCUMENT` operation to retrieve records, you can sort the results of the query using the `ORDER_BY` tag.

*Syntax:*

```
<GET_DOCUMENT ...>
.................
<ORDER_BY>
<Property1 Sort="Ascending|Descending">
<Property2 Sort="Ascending|Descending">
.................
```

```
</ORDER_BY>

</GET_DOCUMENT>
```

*Attributes:*

None

*Example 1:*

The following is an example of a sorting query that returns a list of `LINE_ITEM` instances, sorted by descending order of their create date:

```
<GET_DOCUMENT Name="LINE_ITEM">

<ITEM_ID Value="Muffler" MatchBy="EQUAL"/>

<ORDER_BY>

<CREATION_DATE Sort="Descending"/>

</ORDER_BY>

</GET_DOCUMENT>
```

The response to the above query is a follows:

```
<RESPONSE Status="success" Description="Get Document Successful">

<LINE_ITEM>

<ID Value="LI-1254347890"/>

<ITEM_ID Value="Muffler"/>

<SUPPLIER_ID Value="Synekdo"/>

<PROMISED_TOTAL_QTY Value="75"/>

<CREATION_DATE Value="10/12/2000 15:30:42"/>

<STATUS Value="open"/>

</LINE_ITEM>

<LINE_ITEM>

<ID Value="LI-7778889"/>

<ITEM_ID Value="Muffler"/>

<SUPPLIER_ID Value="dram co."/>

<PROMISED_TOTAL_QTY Value="100"/>

<CREATION_DATE Value="05/11/2000"/>

<STATUS Value="open"/>

</LINE_ITEM>

</RESPONSE>
```

## GROUP_BY

*Description:*

You can use the `GROUP_BY` tag within a `GET_DOCUMENT` operation to sort and select documents based on its properties. The query also returns the count of each aggregated result.

*Syntax:*

```
<GET_DOCUMENT ...>

......

<GROUP_BY>

<PROPERTY_NAME1/>

<PROPERTY_NAME2/>

......

</GROUP_BY>

</GET_DOCUMENT>
```

*Attributes:*

None

*Example 1:*

The following is an example of a sorting query that returns a list of `ORDER_POINT_SHIP_TO_MAP` instances, grouped and sorted by the ORG_ID property.

```
<GET_DOCUMENT Name="ORDER_POINT_SHIP_TO_MAP" ReturnRowCount="true"
MaxRows="5">

<SELECT>

<ORG_ID/>

<SHIP_TO_ID/>

<ORDER_POINT_ID/>

</SELECT>

<GROUP_BY>

<ORG_ID Value="Organization"/>

</GROUP_BY>

<ORDER_BY>

<ORG_ID/>

</ORDER_BY>

</GET_DOCUMENT>
```

The response from the above query is:

```
<RESPONSE TotalRowCount="10" Status="Success">
<ORDER_POINT_SHIP_TO_MAP ExistingDocument="yes" Count="2">
< Organization Value="ORG_1"/>
</ORDER_POINT_SHIP_TO_MAP>
<ORDER_POINT_SHIP_TO_MAP ExistingDocument="yes" Count="8">
< Organization Value="ORG_10"/>
</ORDER_POINT_SHIP_TO_MAP>
<ORDER_POINT_SHIP_TO_MAP ExistingDocument="yes" Count="1">
< Organization Value="ORG_11"/>
</ORDER_POINT_SHIP_TO_MAP>
<ORDER_POINT_SHIP_TO_MAP ExistingDocument="yes" Count="4">
< Organization Value="ORG_111"/>
</ORDER_POINT_SHIP_TO_MAP>
<ORDER_POINT_SHIP_TO_MAP ExistingDocument="yes" Count="4">
< Organization Value="ORG_112"/>
</ORDER_POINT_SHIP_TO_MAP>
</RESPONSE>
```

## QUERY_DOCUMENT_LINK

*Description:*

The `QUERY_DOCUMENT_LINK` tag can be used within a `GET_DOCUMENT` operation to create a table-join between linked documents (declared using the `DOCUMENTLINK` tag. See Chapter 7: "X-Document" for more details).

*Syntax:*

```
<GET_DOCUMENT ...>
<Property1 Value="..."/>
.............
<QUERY_DOCUMENT_LINK Name= "DocumentLinkName">
<Property1 Value="..."/>
<Property2 Value="..."/>
.............
</QUERY_DOCUMENT_LINK>
</GET_DOCUMENT>
```

*Attributes:*

Name (Required): Document link name.

*Example 1:*

To retrieve purchase orders and the corresponding `LINE_ITEMS`, of a particular buyer where at-least one of the line items is in closed state, you can issue the following query:

```
<REQUESTS>
   <GET_DOCUMENT Name="PURCHASE_ORDER">
      <BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
      <QUERY_DOCUMENT_LINK Name="LINE_ITEMS">
         <STATUS Value="Closed"/>
      </QUERY_DOCUMENT_LINK>
      <ORDER_BY>
         <CREATION_DATE/>
      </ORDER_BY>
      <FETCH_DOCUMENT_LINK Name="LINE_ITEMS"/>
   </GET_DOCUMENT>
</REQUESTS>
```

The response of the above query is:

```
<RESPONSES>
   <RESPONSE Status="success" Description="Get Document Successful">
      <PURCHASE_ORDER>
         <ID Value=" PO-9854347890"/>
         <BUYER_ID Value="buyer01"/>
         <PROMISED_DELIVERY_TIME Value="09/17/2000 15:30:42"/>
         <STATE Value="open"/>
         <PIN Value="1"/>
         <INVOICE Value="7000.0"/>
         <MY Value="7000.0"/>
         <MY_ATTRIBUTE_A Value="This is a user attribute"/>
         <MY_ATTRIBUTE_B Value="Another user attribute"/>
         <LINE_ITEMS>
            <LINE_ITEM>
               <ID Value="LI-1254347890"/>
```

```
                    <ITEM_ID Value="Muffler"/>
                    <SUPPLIER_ID Value="Synekdo"/>
                    <PROMISED_TOTAL_QTY Value="75"/>
                    <CREATION_DATE Value="10/12/2000 15:30:42"/>
                    <STATUS Value="open"/>
                 </LINE_ITEM>
                 <LINE_ITEM>
                    <ID Value="LI-7778889"/>
                    <ITEM_ID Value="mem_chip_123"/>
                    <SUPPLIER_ID Value="dram co."/>
                    <PROMISED_TOTAL_QTY Value="100"/>
                    <CREATION_DATE Value="05/11/2000"/>
                    <STATUS Value="closed"/>
                 </LINE_ITEM>
              </LINE_ITEMS>
           </PURCHASE_ORDER>
        </RESPONSE>
</RESPONSES>
```

The 2nd LINE_ITEM instance has a "closed" status.

## NOT_EXIST_QDL

*Description:*

The `NOT_EXISTS_QDL` tag is used within a `GET_DOCUMENT` operation to create a table-join such that only those documents are selected that do not have associated instances in a linked document.

*Syntax:*

```
<GET_DOCUMENT Name="...">
.........
<NOT_EXISTS_QDL Name="{Doc Link Name}"/>
</GET_DOCUMENT>
```

*Attributes:*

Name (Required): Document link name.

*Example 1:*

The following query will retrieve all order point records that do not have entries in the Bill To and Ship To link tables.

```
<GET_DOCUMENT Name="ORDER_POINT">
<STATUS Value="ACTIVE"/>
<OR>
<NOT_EXISTS_QDL Name="SHIP_TO_LIST_LINK"/>
<NOT_EXISTS_QDL Name="BILL_TO_LIST_LINK">
<BT_STATUS Value="ACTIVE"/>
</NOT_EXISTS_QDL>
</OR>
</GET_DOCUMENT>
```

The response from the query is:

```
<RESPONSE Status="Success">
<ORDER_POINT ExistingDocument="yes">
<ID Value="OP06"/>
<ADDRESS_ID Value="addr-771063910853479"/>
<ALLOWS_BACKORDERS Value="false"/>
<CURRENCY_CODE Value="USD"/>
<CUSTOMER_ORG_ID Value="ORG_15"/>
<CUSTOMER_STATUS Value="ACTIVE"/>
<CUST_ORG_FNAME Value="Teradata Technologies"/>
<CUST_ORG_NAME Value="Teradata"/>
<DEFAULT_BILL_TO_ID Value="BT03"/>
<DEFAULT_SHIP_TO_ID Value="SP06"/>
<NAME Value="Royal Teradata"/>
<REQUIRES_ORDER_NUMBER Value="false"/>
<STATUS Value="ACTIVE"/>
</ORDER_POINT>
</RESPONSE>
```

*Example 2:*

The following query will retrieve all organization records that have an associated order point but the order point in turn does not have ship to association:

```
<GET_DOCUMENT Name="ORGANIZATION">
```

```
<QUERY_DOCUMENT_LINK Name="ORDER_POINT_LIST_LINK">
<NOT_EXISTS_QDL Name="SHIP_TO_LIST_LINK">
<QUERY_DOCUMENT_LINK Name="SHIP_TO_INFO_LINK">
<ORG_ID Value="ORG_71"/>
</QUERY_DOCUMENT_LINK>
</NOT_EXISTS_QDL>
</QUERY_DOCUMENT_LINK>
</GET_DOCUMENT>
```

The response from this query is:

```
<RESPONSE Status="Success">
<ORGANIZATION ExistingDocument="yes">
<ID Value="ORG_1"/>
<ADDRESS_ID Value="addr-11063910768266"/>
<ALLOWS_BACKORDERS Value="true"/>
<CURRENCY_CODE Value="USD"/>
<CUSTOMER_STATUS Value="ACTIVE"/>
<DEFAULT_FC_ID Value="FC_2"/>
<FULL_NAME Value="GE Worldwide"/>
<GENERATE_INVOICES Value="true"/>
<IS_REPAIR_VENDOR Value="true"/>
<IS_SINGLE_CORPORATE Value="false"/>
<IS_VRMA_REQUIRED Value="false"/>
<NAME Value="GE"/>
<SELLER_STATUS Value="ACTIVE"/>
<TAX_CODE Value="FEIN"/>
<TAX_ID Value="234T34T23TG45Y2Y"/>
<TRANSFER_ORDERS_BILLABLE Value="false"/>
<URL Value="www.ge.com"/>
</ORGANIZATION>
</RESPONSE>
```

## SELECT + QUERY_DOCUMENT_LINK

*Description:*

The properties of a linked document can be retrieved alongside the property elements of the parent document by using the QUERY_DOCUMENT_LINK and SELECT tags together. An alias can be specified for each selected property in the Query document link and it will be used as the property name in the parent document.

*Syntax:*

```
<GET_DOCUMENT ...>
<QUERY_DOCUMENT_LINK Name= "DocumentLinkName">
<SELECT>
<Property1/>
<Property2 Alias="..."/>
.........
</SELECT>
{Query criteria for linked document}
</QUERY_DOCUMENT_LINK>
</GET_DOCUMENT>
```

*Example 1:*

The following query returns the ITEM_ID (as ITEM) and ORDERED_QTY (as LINE_QTY) fields from the ORDER_LINE_ITEM document and DELIVERY_DATE (as REQ_DELIVERY_DATE) from the ORDER_REQUEST document into the parent, CUSTOMER_ORDER, document.

```
<GET_DOCUMENT Name="CUSTOMER_ORDER" Distinct="true">
<SELECT>
<ID/>
<ORDER_POINT_ID/>
</SELECT>
<QUERY_DOCUMENT_LINK Name="ORDER_LINE_ITEMS">
<SELECT>
<ITEM_ID Alias="ITEM"/>
<ORDERED_QTY Alias="LINE_QTY"/>
</SELECT>
<QUERY_DOCUMENT_LINK Name="ORDER_REQUESTS">
<SELECT>
<DELIVERY_DATE Alias="REQ_DELIVERY_DATE"/>
</SELECT>
```

```
</QUERY_DOCUMENT_LINK>
</QUERY_DOCUMENT_LINK>
<ORDER_BY>
<ID/>
</ORDER_BY>
</GET_DOCUMENT>
```

The response of the above query is:

```
<RESPONSE Status="Success">
<CUSTOMER_ORDER ExistingDocument="yes">
<ID Value="CO-1"/>
<ORDER_POINT_ID Value="OP01"/>
<ITEM Value="151-S"/>
<LINE_QTY Value="6.0"/>
<REQ_DELIVERY_DATE Value="09/23/2003 00:00:00:000"/>
</CUSTOMER_ORDER>
</RESPONSE>
```

## FETCH_DOCUMENT_LINK

*Description:*

The `FETCH_DOCUMENT_LINK` tag can be used within the `GET_DOCUMENT` operation to retrieve linked document instances. In addition, you could specify query criteria to restrict the documents fetched, use `ORDER_BY` tag to sort the linked documents and nest `FETCH_DOCUMENT_LINK` tags to retrieve nested documents.

*Syntax:*

The format of the `FETCH_DOCUMENT_LINK` tag is as follows

```
<GET_DOCUMENT Name="myDoc">
.............
<FETCH_DOCUMENT_LINK Name="documentLinkName">
</GET_DOCUMENT>
```

*Attributes:*

Name (Required): Document to be deleted.

*Example 1:*

The following is an example of fetching document that returns `LINE_ITEMS` and `PROMISED SCHEDULE` Instances along with the `PURCHASE_ORDER`.

```
<REQUESTS>
<GET_DOCUMENT Name="PURCHASE_ORDER">
<ID Value="PO-7854347890" MatchBy="EQUAL"/>
<FETCH_DOCUMENT_LINK Name="LINE_ITEMS">
<!-- LINE_ITEM is the linked document for LINE_ITEMS. Now include
other FETCH_DOCUMENT_LINK tags referencing the document links of
LINE_ITEM.-->
<FETCH_DOCUMENT_LINK Name="SCHEDULES">
<!-- PROMISED_SCHEDULE is the linked document for SCHEDULE. Now,
there is the option to include other FETCH_DOCUMENT_LINK tags
referencing the document links of PROMISED_SCHEDULE-->
<!--You can also use the ORDER_BY tag. The Order by tag should
contain properties of only PROMISED_SCHEDULE document-->
</FETCH_DOCUMENT_LINK>
<!--Your ORDER_BY should contain properties of only LINE_ITEM
document-->
<ORDER_BY Sort="Ascending">
<PROMISED_TOTAL_QTY/>
</ORDER_BY>
</FETCH_DOCUMENT_LINK>
</GET_DOCUMENT>
</REQUESTS>
```

The response to the above request is as follows:

```
<RESPONSES>
<RESPONSE Status="success" Description="Get Document Successful">
<PURCHASE_ORDER >
<ID Value=" PO-7854347890"/>
<BUYER_ID Value="buyer01"/>
<PROMISED_DELIVERY_TIME Value="09/17/2000 15:30:42"/>
<STATE Value="open"/>
<PIN Value="1"/>
<INVOICE Value="7000.0"/>
<MY Value="7000.0"/>
<MY_ATTRIBUTE_A Value="This is a user attribute"/>
```

```
<MY_ATTRIBUTE_B Value="Another user attribute"/>
</PURCHASE_ORDER >
<LINE_ITEMS >
<LINE_ITEM >
<ID Value="LI-1254347890"/>
<ITEM_ID Value="Muffler"/>
<SUPPLIER_ID Value="Synekdo"/>
<PROMISED_TOTAL_QTY Value="75"/>
<CREATION_DATE Value="10/12/2000 15:30:42"/>
<STATUS Value="open"/>
<SCHEDULES >
<PROMISED_SCHEDULE>
<ID Value="PS-994347890"/>
<PROMISED_DATE Value="11/12/2000 11:45:30"/> <PROMISED_QUANTITY
Value="50"/>
</PROMISED_SCHEDULE>
<PROMISED_SCHEDULE>
<ID Value="PS-784347890"/>
<PROMISED_DATE Value="11/15/2000 11:45:30"/>
<PROMISED_QUANTITY Value="25"/>
</PROMISED_SCHEDULE>
</SCHEDULES >
</LINE_ITEM >
</LINE_ITEMS>
</RESPONSE>
</RESPONSES>
```

The `LINE_ITEM` instances are in ascending order of `PROMISED_TOTAL_QTY`.

## GET_DOCUMENT_LINK

*Description:*

The `GET_DOCUMENT_LINK` operation retrieves all the documents related to the current document by the specified document link. The difference between this operation and `FETCH_DOCUMENT_LINK` is that the latter retrieves linked documents only when used within the `GET_DOCUMENT` tags.

*Syntax:*

```
<GET_DOCUMENT_LINK Name="documentName" Link="documentLinkName">
{A Simple or Compound query criteria}
</GET_DOCUMENT_LINK>
```

*Attributes:*

- Name: Document name.
- Link: Document link name.

*Example 1:*

The following queries the document instances, related to the `PURCHASE_ORDER` document instances, of `buyer01` through the `LINE_ITEMS` document link:

```
<GET_DOCUMENT_LINK Name="PURCHASE_ORDER" Link="LINE_ITEMS">
<BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
</GET_DOCUMENT_LINK>
```

The response for a successful execution:

```
<RESPONSE Status="success" Description="Get Document Link
Successful">
<LINE_ITEM>
<ID Value="LI-7778889"/>
<ITEM_ID Value="mem_chip_123"/>
<SUPPLIER_ID Value="dram co."/>
<PROMISED_TOTAL_QTY Value="100"/>
<CREATION_DATE Value="05/11/2000"/>
<STATUS Value="closed"/>
</LINE_ITEM>
<LINE_ITEM>
<ID Value="LI-1254347890"/>
<ITEM_ID Value="Muffler"/>
<SUPPLIER_ID Value="Synekdo"/>
<PROMISED_TOTAL_QTY Value="75"/>
<CREATION_DATE Value="10/12/2000 15:30:42"/>
<STATUS Value="open"/>
</LINE_ITEM>
</RESPONSE>
```

The response for an unsuccessful execution:

```
<RESPONSE Status="error" Description="Document: PURCHASE_ORDER does
not exist"/>
```

## PROCESS_GET_DOCUMENT

*Description:*

The PROCESS_GET_DOCUMENT operation retrieves a set of documents based on query criteria identical to that of the GET_DOCUMENT operation (Please see details on the GET_DOCUMENT operation for further information).However

PROCESS_GET_DOCUMENT retrieves documents in batches with the batch size specified as an attribute. On each batch of records retrieved, a set of actions can be executed conditionally.

This operation is useful for processing high-volume data without running into memory issues; and also for conditional processing of a batch of records.

*Syntax:*

```
<PROCESS_GET_DOCUMENT Name="documentName" BatchSize="noOfRows">
<GET_DOCUMENT_BODY >
<Property1 Value="..." MatchBy="...">
<Property2 Value="..." MatchBy="...">
......
<SELECT>
<PropertyA/>
<PropertyB/>
......
</SELECT>
</GET_DOCUMENT_BODY>
<ON_EACH_BATCH  List="ListVariable" ContinueExpr="...">
<FOR_EACH SelectList="$ListVariable" CurrentElement="...">
</FOR_EACH>
</ON_EACH_BATCH>
</PROCESS_GET_DOCUMENT>
```

*Attributes:*

- Name (Required): The name of the X_Document to be retrieved
- BatchSize (Required): The batch size is indicated here.
- ServiceName (Optional): Service from which the document is retrieved. If not specified, then owner service is taken as default

- List (Required): Variable to which the retrieved batch of records are assigned
- ContinueExpr (Optional): X_Path expression or String value. This expression should evaluate to true for the actions within the `ON_EACH_BATCH` to be executed. The default value is 'true()'

*Example 1:*

The following updates the price incrementally for each batch on the LineItem document instances.

```
<PROCESS_GET_DOCUMENT Name="LineItem1" BatchSize="2">
          <GET_DOCUMENT_BODY>
            <Price Value="8" MatchBy="EQUAL"/>
          </GET_DOCUMENT_BODY>
          <ON_EACH_BATCH List="lines" ContinueExpr="true()">
            <SET Var="newPrice" FromValue="{$newPrice + 1}"/>
            <FOR_EACH SelectList="$lines" CurrentElement="item">
              <MODIFY_DOCUMENT Name="LineItem" AssignToVar="result">
                <DOCUMENT_CONTEXT>
                  <ID Value="{$item/ID/@Value}"/>
                </DOCUMENT_CONTEXT>
                <UPDATE_PROPERTIES>
                  <Price Value="{$newPrice}"/>
                </UPDATE_PROPERTIES>
              </MODIFY_DOCUMENT>
            </FOR_EACH>
          </ON_EACH_BATCH>
        </PROCESS_GET_DOCUMENT>
```

The response for a successful execution:

```
The price one the first 2 records is updated to 9. The next 2 to 10
and so-on.
```

The response for an unsuccessful execution:

```
<RESPONSE Status="error" Description="Document: does not exist"/>
```

# Smart XCommands

Smart Xcommands are the advanced form of basic Xcommands - GET_DOCUMENT, ADD_DOCUMENT, REMVOE_DOCUMENT, MODIFY_DOCUMENT. These are used to peform DML operations SELECT, INSERT, DELETE and UPDATE respectively on a database table. These smart xcommands have the additional capability to deal with System fields while acting upon Master tables.

## ADD_DOC_SMART

*Description:*

ADD_DOC_SMART auto-fills system fields EntityState with 'ACTIVE', CreationDate and LastModifiedDate with current date.

*Syntax:*

```
<ADD_DOC_SMART Name="documentName">
 <Property1 Value="...">
 <Property2 Value="...">
 ......
</ADD_DOC_SMART>
```

*Attributes:*

All ADD_DOCUMENT xcommand attributes are applicable on ADD_DOC_SMART.

*Example:*

```
<ADD_DOC_SMART Name="LINE_ITEM">
 <itemID Value="IT-101"/>
 <ItemName Value="Muffler"/>
 <ItemCategoryCode Value="FROZEN_GOODS"/>
</ADD_DOC_SMART>
```

## GET_DOC_SMART

*Description:*

GET_DOC_SMART retrieve records which are not in DELETED state (EntityState = DELETED) or where EntityState IS NULL.

*Syntax:*

```
<GET_DOC_SMART Name="documentName">
 <Property1 Value="..." MatchBy="...">
 <Property2 Value="..." MatchBy="...">
 ......
 <SELECT>
  <PropertyA/>
  <PropertyB/>
  ......
 </SELECT>
</GET_DOC_SMART>
```

*Attributes:*

All GET_DOCUMENT xcommand attributes are applicable on GET_DOC_SMART.

*Example:*

```
<GET_DOC_SMART Name="LINE_ITEM">
 <ITEM_ID Value="Muffler" MatchBy="EQUAL"/>
</GET_DOC_SMART>
```

## MODIFY_DOC_SMART

*Description:*

MODIFY_DOC_SMART auto-fills LastModifiedDate and LastModifiedBy system fields.

*Syntax:*

```
<MODIFY_DOC_SMART Name="documentName">
 <DOCUMENT_CONTEXT>
  {A Simple or Compound query criteria}
 </DOCUMENT_CONTEXT>
 <UPDATE_PROPERTIES>
  <PropertyName1 Value="value1"/>
  ............
 </UPDATE_PROPERTIES>
</MODIFY_DOC_SMART>
```

*Attributes:*

All MODIFY_DOCUMENT xcommand attributes are applicable on MODIFY_DOC_SMART.

*Example:*

```
<MODIFY_DOC_SMART Name="PURCHASE_ORDER">
 <DOCUMENT_CONTEXT>
  <BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
 </DOCUMENT_CONTEXT>
 <UPDATE_PROPERTIES>
  <STATE Value="hold"/>
 </UPDATE_PROPERTIES>
</MODIFY_DOC_SMART>
```

## REMOVE_DOC_LOGICAL

*Description:*

REMOVE_DOC_LOGICAL refers to HardDelete setting defined at document design level . Default is to perform soft delete (EntityState = DELETED). REMOVE_DOC_LOGICAL xcommand is only applicable for Master staging tables.

*Syntax:*

```
<REMOVE_DOC_LOGICAL Name="doc-name">
 {A simple or compound query criteria}
</REMOVE_DOC_LOGICAL>
```

*Attributes:*

All REMOVE_DOCUMENT xcommand attributes are applicable on REMOVE_DOC_LOGICAL.

*Example:*

```
<REMOVE_DOCUMENT Name="PURCHASE_ORDER">
 <BUYER_ID Value="buyer01" MatchBy="EQUAL"/>
</REMOVE_DOCUMENT>
```

# SQL Operations

## SQL_FN

*Description:*

The `SQL_FN` tag can be used within the select clause of a `GET_DOCUMENT` query to perform selected mathematical functions on a document property. The supported functions are:

- SUM (Summation of property values)
- COUNT (Count of all the records that match a certain criteria. Note that it is different from the ReturnRowCount and RowCount attributes)
- MIN (Minimum among all the property values)
- MAX (Maximum among all the property values)

SQL functions can take as arguments document properties, a constant value and other SQL functions. Arithmetic operators can also be specified to compute the result of the SQL function.

You can extend the com.teradata.xcore.too.sql.SQLFn class to implement a SQL function.

*Syntax:*

```
<SQL_FN Name="{Function}" Alias="{Query Result Name}">

<PROP Name="{Property Name}"/>

</SQL_FN>
```

*Attributes:*

- Name (Required): SUM/COUNT/MIN/MAX
- Alias (Required): Property to which the query result is assigned.

*Example 1:*

The following example shows how the `SUM` function can be used in multiple ways:

```
<GET_DOCUMENT Name="INVENTORY">

<SELECT>

<ITEM_ID/>

<LOCATION_ID/>

<SQL_FN Name="SUM" Alias="SUM_IN_TRANSIT">

<PROP Name="IN_TRANSIT"/>
```

```
/SQL_FN>
<!-Expr below: InTransit + OnOrder - Damaged -->
<SQL_FN Name="SUM" Alias="EXPECT_SUM">
<MINUS>
ADD>
<PROP Name="IN_TRANSIT"/>
<PROP Name="ON_ORDER"/>
</ADD>
<PROP Name="DAMAGED"/>
</MINUS>
</SQL_FN>
<!-Expr:(2 x InTransit)/2 + OnOrder - Damaged -->
<SQL_FN Name="SUM" Alias="COMPLEX_SUM">
<MINUS>
<ADD>
<DIV>
<MULT>
<CONSTANT Value="2"/>
<PROP Name="IN_TRANSIT"/>
</MULT>
<CONSTANT Value="2"/>
</DIV>
<PROP Name="ON_ORDER"/>
</ADD>
<PROP Name="DAMAGED"/>
</MINUS>
</SQL_FN>
</SELECT>
<ITEM_ID Value="151-S"/>
<OR>
<LOCATION_ID Value="FC_1"/>
<LOCATION_ID Value="FC_2"/>
```

```
<LOCATION_ID Value="DC_1"/>
</OR>
<GROUP_BY>
<ITEM_ID/>
<LOCATION_ID/>
</GROUP_BY>
<ORDER_BY>
<ALIAS Name="COMPLEX_SUM" Sort="Descending"/>
</ORDER_BY>
</GET_DOCUMENT>
```

The response from the above query is:

```
<RESPONSE Status="Success">
<INVENTORY ExistingDocument="yes" Count="1">
<SUM_IN_TRANSIT Value="5000.0"/>
<EXPECT_SUM Value="6900.0"/>
<COMPLEX_SUM Value="6900.0"/>
<ITEM_ID Value="151-S"/>
<LOCATION_ID Value="DC_1"/>
</INVENTORY>
<INVENTORY ExistingDocument="yes" Count="2">
<SUM_IN_TRANSIT Value="5000.0"/>
<EXPECT_SUM Value="6800.0"/>
<COMPLEX_SUM Value="6800.0"/>
<ITEM_ID Value="151-S"/>
<LOCATION_ID Value="FC_1"/>
</INVENTORY>
</RESPONSE>
```

*Example 2:*

The following example shows how the `SUM` and `MAX` functions can be used:

```
<GET_DOCUMENT Name="INVENTORY" StartAtRow="0" MaxRows="2"
ReturnRowCount="yes">
<SELECT>
<ITEM_ID/>
```

```
<LOCATION_ID/>
<SQL_FN Name="SUM" Alias="QTY">
<PROP Name="UNRESTRICTED"/>
</SQL_FN>
<SQL_FN Name="MAX" Alias="LAST_UPDATE_TIME">
<PROP Name="LAST_UPDATE_TIME"/>
</SQL_FN>
</SELECT>
<ITEM_ID Value="1" MatchBy="STARTS_WITH"/>
<GROUP_BY>
<ITEM_ID/>
<LOCATION_ID/>
</GROUP_BY>
</GET_DOCUMENT>
```

The response from the above query is:

```
<RESPONSE TotalRowCount="4" Status="Success">
<INVENTORY ExistingDocument="yes" Count="3">
<QTY Value="200.0"/>
<LAST_UPDATE_TIME Value="01/19/2004 20:51:08:000"/>
<ITEM_ID Value="151-S"/>
<LOCATION_ID Value="FC_1"/>
</INVENTORY>
<INVENTORY ExistingDocument="yes" Count="3">
<QTY Value="900.0"/>
<LAST_UPDATE_TIME Value="01/16/2004 09:34:55:000"/>
<ITEM_ID Value="1C6936G05"/>
<LOCATION_ID Value="FC_1"/>
</INVENTORY>
</RESPONSE>
```

*Example 3:*

The following example shows how the `COUNT` function is used:

```
<GET_DOCUMENT Name="INVENTORY" >
```

```
<SELECT>
<SQL_FN Name="COUNT" Alias="TOTAL_COUNT"/>
</SELECT>
<ITEM_ID Value="1" MatchBy="STARTS_WITH"/>
</GET_DOCUMENT>
```

The response from the above query is:

```
<RESPONSE Status="Success">
<INVENTORY ExistingDocument="yes" Count="3"/>
</RESPONSE>
```

## Scalar Functions

*Description:*

For performing column operations in the where clause of a SQL query, the 'ScalarFn' attribute can be used on a property name in a GET_DOCUMENT operation. The attribute operates on a single column and supports the following functions:

- upperCase: To enable upper case conversion of a column
- lowerCase: To enable lower case conversion of a column
- char: To convert a column to char
- float: To convert column to numeric value
- double: To convert column to numeric value

*Syntax:*

```
<GET_DOCUMENT Name="CAR">
.............
<Property1 Value="Value" MatchBy="Operator" ScalarFn="function"/>
.............
</GET_DOCUMENT>
```

*Example 1:*

```
<GET_DOCUMENT Name="PARENT">
<NAME Value="Computer" MatchBy="LIKE" ScalarFn="upperCase"/>
</GET_DOCUMENT>
```

Will result in query with `upper(NAME) LIKE 'Computer%'` in SQL.

*Example 2:*

```
<GET_DOCUMENT Name="CAR">
<MODEL Value="TT"/>
```

```
<MSRP CompareProperty="PRICE" MatchBy="NOT_EQUAL" ScalarFn="char"/>
</GET_DOCUMENT>
```

*Example 3:*

```
<GET_DOCUMENT Name="CAR">
<MODEL Value="TT"/>
<MSRP CompareProperty="PRICE" MatchBy="GREATER_EQUAL"
ScalarFn="float"/>
</GET_DOCUMENT>
```

*Example 4:*

```
<GET_DOCUMENT Name="CAR">
<MODEL Value="TT"/>
<PRICE CompareProperty="MSRP" MatchBy="GREATER" ScalarFn="double"/>
</GET_DOCUMENT>
```

## EXECUTE_SQL_QUERY

*Description:*

You can perform a raw SQL query on the database and get the results of the query in a XML format. The calling application is responsible for issuing syntactically correct SQL statement. You can also issue paginated raw sql queries.

*Syntax:*

```
<EXECUTE_SQL_QUERY Value= "sql-string" StartAtRow=".."
ReturnRowCount="yes"} {MaxRows=".."}
{GetNullTags="True|Yes|False|No"}>
</EXECUTE_SQL_QUERY >
```

*Attributes:*

- Value (Required): SQL query.
- StartAtRow (Optional): Beginning row number.
- MaxRows (Optional): Maximum number of rows to be returned.
- ReturnRowCount (Optional): 'yes/no'. If 'yes', then the response will contain another attribute 'TotalRowCount' specifying the total number of records that qualified the query string. Default is 'no'.
- GetNullTags (Optional): 'yes/no'. If 'yes', then empty tags are returned for null values; default is 'no'.
- AssignToVar (Optional): Variable to which the query result is assigned.

*Example 1:*

```
<EXECUTE_SQL_QUERY Value="Select count(*) As COUNT from
PURCHASE_ORDER"/>
```

Response from the above query is:

```
<RESPONSE Status="Success">
<SQL_RESULT>
<COUNT Value="45"/>
</SQL_RESULT>
</RESPONSE>
```

*Example 2:*

```
<EXECUTE_SQL_QUERY Value="Select * FROM CUSTOMERORDER"
GetNullTags="yes" MaxRows="1"}/>
```

Response from the above query is:

```
<RESPONSE Status="success" >
<SQL_RESULT>
<ORDERID Value="CO1"/>
<CREATEDATE/>
<CUSTID Value="jdoe"/>
<ID Value="1"/>
</SQL_RESULT>
</RESPONSE>
```

## EXECUTE_SQL_DML

*Description:*

You can execute a raw SQL data manipulation language (DML) statement on the database and get the results of the query in a XML format. The calling application is responsible for issuing syntactically correct SQL statement.

The response contains an 'UpdateRowCount' attribute that is assigned the number of rows updated by the SQL query.

*Syntax:*

```
<EXECUTE_SQL_DML Value= "sql-string" AssignToVar="myVar">
</EXECUTE_SQL_DML>
```

*Attributes:*

- Value (Required): SQL DML statement.
- AssignToVar (Optional): Variable to which the query result is assigned.

*Example 1:*

```
<EXECUTE_SQL_DML Value="DELETE FROM WFL_ATV_EVENTS WHERE TEMPLATE =
'TEST'" AssignToVar="response"/>
```

The response from the above query is:

```
<RESPONSES>
   <RESPONSE Status="success" UpdateRowCount="1000">
   </RESPONSE>
</RESPONSES>
```

## EXECUTE_BATCH_SQL_DML

*Description:*

EXECUTE_BATCH_SQL_DML xcommand is raw SQL execution command which bypasses the MDM Xcore layer.This will not support any configuration achieved using any Xcore Xcommands i.e. GET_DOC, ADD_DOC such as data authorization (RLS/CLS) etc. Secondly DO_DB_PERSIST provides many integral operations such as RI check, xml validation, sql validation, audit trail handling, raise alert, execute persist rules (sql_insert, sql_update) etc which will not be achieved using EXECUTE_BATCH_SQL_DML .

**Note:** Each <PARAMS> tag represents a row and <PARAM> signifies the column in that row

Insert Example:

```
<EXECUTE_BATCH_SQL_DML Value="INSERT INTO MST_ATTRIBUTE_SET
(ATTRIBUTE_SET_ID,UPDTD_LOG_TABLE_NAME) VALUES (?,?)">
  <PARAMS>
   <PARAM Type="string" Value="att2"/>
   <PARAM Type="string" Value="log_upd_table2"/>
  </PARAMS>
  <PARAMS>
   <PARAM Type="string" Value="att3"/>
   <PARAM Type="string" Value="log_upd_table3"/>
  </PARAMS>
</EXECUTE_BATCH_SQL_DML>
```

Update Example:

```
<EXECUTE_BATCH_SQL_DML Value="UPDATE MST_ATTRIBUTE_SET SET NAME=?
WHERE ATTRIBUTE_SET_ID=?">
  <PARAMS>
   <PARAM Type="string" Value="att2_name"/>
   <PARAM Type="string" Value="att2"/>
  </PARAMS>
```

```
  <PARAMS>
   <PARAM Type="string" Value="att3_name"/>
   <PARAM Type="string" Value="att3"/>
  </PARAMS>
</EXECUTE_BATCH_SQL_DML>
```

## EXECUTE_SQL_DDL

*Description:*

You can execute a raw SQL data definition language (DDL) statement on the database and get the results of the query in a XML format. The calling application is responsible for issuing syntactically correct SQL statement.

The response contains an 'UpdateRowCount' attribute that is assigned the number of rows updated by the SQL query.

*Syntax:*

```
<EXECUTE_SQL_DDL Value= "sql-string" AssignToVar="myVar">
</EXECUTE_SQL_DDL>
```

*Attributes:*

- Value (Required): SQL DML statement.
- AssignToVar (Optional): Variable to which the query result is assigned.

*Example 1:*

```
<EXECUTE_SQL_DDL Value=" ALTER TABLE ECS.WFL_ATV_EVENTS ADD (TEST
VARCHAR2(32))" AssignToVar="response"/>
```

The response from the above query is:

```
<RESPONSES>
   <RESPONSE Status="success" UpdateRowCount="0">
   </RESPONSE>
</RESPONSES>
```

## EXEC_PROC

*Description:*

Using the `EXEC_PROC` operation, you can execute a database procedure and get the results of the query in a XML format. The calling application is responsible for issuing syntactically correct SQL statement.

**Note:** To use this tag, ensure that your X-Service file has the following configuration:

The extension rule file 'bcm_rule_extension.xml' is specified. For e.g.:

```
<extensionFiles>
```

```
<extensionFile Name="com/teradata/bcm/xcore/serverutil/xml/
bcm_rule_extension.xml"/>

</extensionFiles>
```

The <extensionFiles> block occurs before the <ruleDefnFiles> block. For e.g.:

```
<dom-config>

......

<extensionFiles>

......

</extensionFiles>

<ruleDefnFiles>

......

</ruleDefnFiles>

......

</dom-config>
```

*Syntax:*

```
<EXEC_PROC Name="procName" AssignToVar="myVar">

<ARGUMENT1 Type="IN/OUT/IN_OUT" Value="value" DataType="string"/>

...

</EXEC_PROC >
```

*Attributes:*

- Name (Required): Procedure name.
- AssignToVar (Optional): Variable to which the query result is assigned.

## Child Tags/Attributes

You have the option to pass one or more arguments to the procedure, each having the following attributes:

- Type (Required): `IN` indicates that the argument is an input to the procedure; `OUT` indicates that the procedure returns this argument; `IN_OUT` indicates that this argument is both an input to and an output from the procedure
- Value (Optional): The value of the argument (not required if it is an `OUT` type)
- DataType (Optional): Type of the argument; string is taken as the default type.

*Example 1:*

In the following example, a procedure 'FIND_CALENDAR_WO_ENTRY' is executed and the output value is used to raise an event:

```
<EXEC_PROC Name="FIND_CALENDAR_WO_ENTRY" AssignToVar="response">
```

```
<STATUS Type="OUT" DataType="string"/>
</EXEC_PROC>
<RAISE_EVENT Name="CalendarValidationFinished">
<STATUS Value="{$response/STATUS/@Value}"/>
</RAISE_EVENT>
```

## EXEC_PARALLEL

*Description:*

Command to execute multiple commands concurrently.

*Syntax:*

```
<EXEC_PARALLEL Mode="Sync" Transaction="true">
   <Command1 ../>
   <Command2 ../>
   <Command3 ../>
   …..
   <Commandn ../>
<EXEC_PARALLEL>
```

*Attributes:*

- Mode: Attribute Mode can be either Sync, Async, OrNode. Default value is "Sync".
  - Sync Mode : The command will wait for all the subtasks to complete before returning the complete response.
  - Async Mode : The command will not wait for the subtasks to complete, will just invoke the requests and then return the control to the caller.
  - OrNode Mode: This is used for supporting the Fork Or Workflow Node. In this mode the command will wait for one of the subtasks to complete and then will return the control to the caller.
- Transaction: : Attribute Transaction can be true or false. Default value is "true". In the case of Transaction value passed as "true", all the subtasks will be executed in one Transaction Mode. So if there is any error in any of the sub tasks, all the sub tasks DB operations are rolled back.Inorder for the transaction to commit, all the subtasks should finish their execution without any errors. In the case of Transaction value set to false, each subtasks are executed in their own transaction and will not affect the other subtask's transaction

*Example:*

```
<EXEC_PARALLEL Mode="Sync" Transaction="true">>
<REQUEST Name="persistTimeData" ServiceName="BPE_META/>
<REQUEST Name="persistSyncUp" ServiceName="BPE_META"/>
                <ADD_DOCUMENT Name="Customer"
ServiceName="BCM_MASTER">
```

```
<Id Value="1"/>
<Description Value="2"/>
</ADD_DOCUMENT>
</EXEC_PARALLEL>
```

In this example, all the three input subtasks are executed in one transaction with the command waiting for all the tasks to complete before returning the response to the caller of this command.

# XML Manipulation Operations

## SET

*Description:*

`SET` assigns the value of the RhsExpression to the LhsExpression.

*Syntax:*

```
<SET LhsExpression RhsExpression/>
```

*Attributes:*

- Var: Untyped variable which `SET` operates upon.
- DocVar: Document variable which `SET` operates upon.
- Property: Any element of the document represented by `DocVar`.
- Attribute: Any attribute of the document represented by `DocVar`.
- FromDocVar: Document that `SET` refers to.
- FromVar: Untyped variable that `SET` refers to.
- FromValue: X-Path expression or String value.
- FromSelect: Untyped variable that `SET` refers to.
- FromValueList: Deprecated.

When you are using `SET`, combinations of these attributes can be used to form the Lhs or Rhs Expressions, depending on the context of the `SET` statement.

*Example 1:*

The following statement creates an un-typed variable called myVar and assigns it the value of ` 50 '.

```
<SET Var="myQuantity" FromValue="50"/>
```

```
LhsExpression: Var="myQuantity"
```

```
RhsExpression: FromValue="50"
```

*Example 2:*

The following statement sets the `PROMISED_TOTAL_QTY` property of the `LINE_ITEM` instance referenced by the variable myDoc to the value of the variable myQuantity.

```
<SET DocVar="myDoc" Property="PROMISED_TOTAL_QTY"
FromValue="{$myQuantity}"/>

LhsExpression: DocVar="myDoc" Property=REMOVE_DOCUMENTY"

RhsExpression: FromValue="{$myQuantity}
```

*Example 3:*

The following statement sets the variable liQty with the value of `ORDERED_QTY` property of the `LINE_ITEM` instance referenced by the variable liDoc.

```
<SET Var="liQty" FromValue="{$liDoc/ORDERED_QTY/@Value}"/>

LhsExpression: Var="liQty"

RhsExpression: FromValue="{$liDoc/ORDERED_QTY/@Value}"
```

**Note:** The DocVar/Property, DocVar/Attribute combinations during SET operation cannot be replaced by equivalent X-Path value semantics.

*Example 4:*

The following statement uses `SET` tag to set an attribute on an already existing document instance.

```
<SET DocVar="myDoc" Attribute="Comment" FromValue="Test"/>

LhsExpression: DocVar="myDoc" Attribute="Comment"

RhsExpression: FromValue="Test"
```

## SET_PROPS

*Description:*

`SET_PROPS` sets the child XML elements on a DocVar object or on each DocVar object present in a list. This is functionally same as SET statement with DocVar and Property attributes.

`SET_PROPS` should be used when more than one property needs to be set on the same parent element. Also, while setting each property, the optional attribute `Overwrite' can be used to specify whether that property should be overwritten or not (by default, it is true).

*Syntax:*

```
<SET_PROPS Select="$myVar">

<Property1 LhsExpression {Overwrite="true/false"}/>

<Property2 LhsExpression {Overwrite="true/false"}/>

..............

<Propertyn LhsExpression {Overwrite="true/false"}/>

</SET_PROPS>
```

*Attributes:*

- Select: Variable on which action is performed.
- SelectList: List of XML form objects on which the action is performed.

*Example 1:*

The following statement will update only the `PRICE` and `LAST_UPDATED` properties of a `BOOK` document (represented by the variable myBook).

```
<SET_PROPS Select="$myBook">
<AUTHOR Value="John Doe" Overwrite="false"/>
<PRICE Value="15.00"/>
<LAST_UPDATED Value="{date()}" Overwrite="yes"/>
</SET_PROPS>
```

### REMOVE_ATTRIBUTE

*Description:*

This statement is used to remove attributes from XML elements, which may be set by the `SET` action statement.

*Syntax:*

```
<REMOVE_ATTRIBUTE Name="attribName" DocVar="docVariable"/>
```

*Attributes:*

- DocVar (Required): Document variable on which action is performed.
- Name (Required): Name of the attribute that should be removed from DocVar.

*Example 1:*

The following statement will remove the Comment attribute on the document instance represented by `myDoc`:

```
<REMOVE_ATTRIBUTE DocVar="myDoc" Name="Comment"/>
```

## TO_DOCVAR

*Description:*

`TO_DOCVAR` statement is used to build an XML document in memory. `TO_DOCVAR` tags can embed other action statements while forming XML to build complex XMLs in memory; the action statements will be executed while building the XML DocVar.

*Syntax:*

```
<TO_DOCVAR AssignToVar="varName">
<XML Document>
</TO_DOCVAR>
```

*Attributes:*

AssignToVar: Variable that will store the XML built by the statement.

*Example 1:*

This example shows how to build a simple XML document (bookDoc) and print it on the console.

Input:

```
<TO_DOCVAR AssignToVar="bookDoc">
<BOOK>
<ID Value="BK-01"/>
<AUTHOR Value="J R R Tolkein"/>
<TITLE Value="Fellowship of the Ring"/>
</BOOK>
</TO_DOCVAR>
<PRINTLN Value="Printing bookDoc:"/>
<PRINTLN Value="{$bookDoc}"/>
```

Output:

Printing bookDoc:

```
<CUSTOMER_ORDER>
<ID Value="CO-01"/>
</CUSTOMER_ORDER>
```

*Example 2:*

This example shows how `TO_XML` action statement (See the next section for details on TO_XML) is used within `TO_DOCVAR` to build a complex XML - DocVar messageDoc:

Input:

```
<TO_DOCVAR AssignToVar="messageDoc">
<MESSAGE_DATA>
<CO_ID Value="{$reqDoc/CUSTOMER_ORDER/ID/@Value}"/>
<HEADER>
<FROM Value="abc"/>
<TO Value="xyz"/>
</HEADER>
<BODY>
<TO_XML SelectList="$reqDoc/CUSTOMER_ORDER/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM">
</BODY>
</MESSAGE_DATA>
```

```
</TO_DOCVAR>
<PRINTLN Value="Printing messageDoc document :"/>
<PRINTLN Value="{$messageDoc}"/>
```

Output:

Printing messageDoc document:

```
<MESSAGE_DATA>
<CO_ID Value="CO-01"/>
<HEADER>
<FROM Value="abc"/>
<TO Value="xyz"/>
</HEADER>
<BODY>
<ORDER_LINE_ITEM>
<ID Value="LI-01"/>
<ITEM_ID Value="CPU"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-02"/>
<ITEM_ID Value="MONITOR"/>
</ORDER_LINE_ITEM>
</BODY>
</MESSAGE_DATA>
```

*Example 3:*

This example shows an X-Rule that creates a complex XML using `TO_DOCVAR` tags with multiple action statements embedded inside.

```
<DEFINE_METHOD Name="createOrderXml">
<RULE>
<ACTION>
<TO_DOCVAR AssignToVar="thisReturn">
<RESPONSE>
<CUSTOMER_ORDER>
<ID Value="2"/>
<REPEAT Count="int($thisParam/NUM_LINES/@Value)"
```

```
IndexVar="currIndex">
<SET Var="liId" FromValue="{concat(`LI_',
string(currIndex))}"/>
<TO_DOCVAR AssignToVar="liDoc">
<ORDER_LINE_ITEM>
<ID Value="{$liId}"/>
</ORDER_LINE_ITEM>
</TO_DOCVAR>
<TO_XML DocVar="liDoc"/>
</REPEAT>
</CUSTOMER_ORDER>
</RESPONSE>
</TO_DOCVAR>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

Output:

```
<RESPONSE>
<CUSTOMER_ORDER>
<ID Value="2"/>
<ORDER_LINE_ITEM>
<ID Value="LI_0"/>
</ORDER_LINE_ITEM>
........
........
<ORDER_LINE_ITEM>
<ID Value="LI_4"/>
</ORDER_LINE_ITEM>
</CUSTOMER_ORDER>
</RESPONSE>
```

In the above example, the `REPEAT` action statement was executed while forming the `TO_DOCVAR`. To prevent the action statement(s) from executing inside `TO_DOCVAR`, the optional attribute `IsLiteral' should be set to `true' (default is false).

*Example:*

```
<TO_DOCVAR AssignToVar="request" IsLiteral="true">
<REPEAT Count="int($thisParam/NUM_LINES/@Value)"
IndexVar="currIndex">
<SET Var="liId" FromValue="{concat(`LI_', string(currIndex))}"/>
</REPEAT>
</TO_DOCVAR>
```

This will construct the REPEAT XML form without executing it.

## TO_XML

*Description:*

TO_XML is an action statement that can be used either by the TO_DOCVAR tag or independently to create the dynamic content for a command (REQUEST, ADD_DOCUMENT etc.). TO_XML only operates on existing document variables or lists, and its output is always added as a child to the enclosing tag.

*Syntax:*

```
<TO_XML DocVar="docvarName"/>
```

OR

```
<TO_XML SelectList="listName"/>
```

*Attributes:*

- DocVar: The document variable used to build the dynamic content
- SelectList: The document list used to build the dynamic content

*Example 1:*

Assume that the DocVar `reqDoc' points to the document instance:

```
<CUSTOMER_ORDER>
<ID Value="CO-01"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-01"/>
<ITEM_ID Value="CPU"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-02"/>
<ITEM_ID Value="MONITOR"/>
</ORDER_LINE_ITEM>
```

```
</ORDER_LINE_ITEMS>

</CUSTOMER_ORDER>
```

The following statements show how to use `TO_XML` to pass the `reqDoc` variable to the processCustomerOrder command:

```
<REQUEST Name="processCustomerOrder">

<TO_XML DocVar="coDoc"/>

</REQUEST>
```

This is equivalent to:

```
<REQUEST Name="processCustomerOrder">

<CUSTOMER_ORDER>

<ID Value="CO-01"/>

<ORDER_LINE_ITEMS>

<ORDER_LINE_ITEM>

<ID Value="LI-01"/>

<ITEM_ID Value="CPU"/>

</ORDER_LINE_ITEM>

<ORDER_LINE_ITEM>

<ID Value="LI-02"/>

<ITEM_ID Value="MONITOR"/>

</ORDER_LINE_ITEM>

</ORDER_LINE_ITEMS>

</CUSTOMER_ORDER>

</REQUEST>
```

*Example 2:*

The following statements demonstrate the use `TO_XML` with `SelectList` attribute (the variable reqDoc is the same as for Example 1).

```
<REQUEST Name="processLineItems">

<TO_XML SelectList="$reqDoc/CUSTOMER_ORDER/ORDER_LINE_ITEMS/
LINE_ITEM"/>

</REQUEST>
```

This is equivalent to:

```
<REQUEST Name="processLineItems">

<ORDER_LINE_ITEM>
```

```
<ID Value="LI-01"/>
<ITEM_ID Value="CPU"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-02"/>
<ITEM_ID Value="MONITOR"/>
</ORDER_LINE_ITEM>
</REQUEST>
```

### APPEND_TO_XML

*Description:*

This action will append new XML elements to a parent element, and hence can be used for extending XMLform structures.

`APPEND_TO_XML` can also be used without referencing a document. In this case, the XML elements under `APPEND_TO_XML` are added to the XMLform object referenced to by the tag above.

*Syntax:*

```
<APPEND_TO_XML DocVar="docName"/>
<CHILD_A/>
<CHILD_B/>
........
</APPEND_TO_XML>
```

*Attributes:*

DocVar: The document variable to append child XML elements to.

*Example 1:*

Input:

`$thisReturn` initially refers to:

```
<RESPONSE/>
```

`$otherTags` referred to:

```
<OTHER_TAGS/>


<APPEND_TO_XML DocVar="thisReturn">
<CHILD_A/>
<CHILD_B/>
```

```
<TO_XML Select="$OtherTags"/>
</APPEND_TO_XML>
```

Output:

After the `APPEND_TO_XML` operation, `thisReturn` would be:

```
<RESPONSE>
<CHILD_A/>
<CHILD_B/>
<OTHER_TAGS/>
</RESPONSE>
```

*Example 2:*

Input:

<TO_DOCVAR AssignToVar="doc"><DOC>

```
<FOR_EACH SelectList="$list/ITEM" CurrentElement="item">
<IF_TEST Test="$item/PRICE > 10">
<THEN>
<APPEND_TO_XML>
<TO_XML DocVar="item"/>
</APPEND_TO_XML>
</THEN>
</IF_TEST>
</FOR_EACH>
</DOC>
</TO_DOCVAR>
```

Output:

After the `APPEND_TO_XML` operation, 'doc' would be:

```
<DOC>
<ITEM>
<ID Value="xxx"/>
<PRICE Value="50"/>
</ITEM>
</DOC>
```

## SET_NAME

*Description:*

The `SET_NAME` action tag changes the name of the root tag referred to by a given variable to a different name.

*Syntax:*

```
<SET_NAME Value="{$docRoot}" FromValue="NEW_NAME"/>
```

*Attributes:*

- Value: Document variable on which action is performed.
- FromValue: Name that the root tag should be changed to.

*Example:*

Input:

```
<TO_DOCVAR AssignToVar="myDoc">
<OLD_NAME>
<CHILDREN/>
</OLD_NAME>
</TO_DOCVAR>
<SET_NAME Value="{$myDoc}" FromValue="NEW_NAME"/>
<PRINTLN Value="{$myDoc}"/>
```

Output:

```
<NEW_NAME>
<CHILDREN/>
</NEW_NAME>
```

## SET_TEXT

*Description:*

The `SET_TEXT` action sets the text content for a given tag.

*Syntax:*

```
<SET_TEXT Value="{$docVar}" FromValue="text content"/>
```

*Attributes:*

- Value: Document variable on which action is performed.
- FromValue: The text content that is set.

*Example 1:*

Input:

```
<TO_DOCVAR AssignToVar="myDoc">
```

```
<DOC_ROOT/>
</TO_DOCVAR>
<SET_TEXT Value="{$myDoc}" FromValue="This is the text content."/>
<PRINTLN Value="{$myDoc}"/>
```

Output:

```
<DOC_ROOT>
This is the text content.
</DOC_ROOT>
```

## GET_TEXT

*Description:*

The `GET_TEXT` action retrieves the text content for a given tag.

*Syntax:*

```
<GET_TEXT FromValue="{value}" AssignToVar="var"/>
```

*Attributes:*

- FromValue: Document variable for which the text content is retrieved.
- AssignToVar: Variable to which the text content is assigned.

*Example 1:*

Input:

```
<TO_DOCVAR AssignToVar="myDoc">
<DOC_ROOT>
This is the text content.
</DOC_ROOT>
</TO_DOCVAR>
<GET_TEXT FromValue="{$myDoc}" AssignToVar="myVar"/>
<PRINTLN Value="{$myVar}"/>
```

Output:

```
This is the text content.
```

## SET_CHILD

*Description:*

The `SET_CHILD` action removes existing children of the same name and adds the new child.

*Syntax:*

```
<SET_CHILD Value="{$docVar}" FromValue="{$childVar}"/>
```

*Attributes:*

- Value: Document variable on which action is performed.
- FromValue: Variable that references the child document.

*Example 1:*

Input:

`$parentDoc` refers to:

```
<DOC_ROOT>
<A_CHILD/>
<B_CHILD>
<EXISTING_B_CHILD1/>
</B_CHILD>
<B_CHILD>
<EXISTING_B_CHILD2/>
</B_CHILD>
</DOC_ROOT>
```

`$childDoc` refers to:

```
<B_CHILD>
<NEW_B_CHILD/>
</B_CHILD>
<SET_CHILD Value="{$doc}" FromValue="{$child}"/>
```

Output:

`$doc` refers to:

```
<DOC_ROOT>
<A_CHILD/>
<B_CHILD>
<NEW_B_CHILD/>
</B_CHILD>
</DOC_ROOT>
```

## ADD_CHILDREN

*Description:*

The `ADD_CHILDREN` statement is used to add one or more documents specified by the X-Path expression to the document Variable 'parentDoc'.

*Syntax:*

```
<ADD_CHILDREN DocVar="parentDoc" FromValue ="{X-PathExpr}" />
```

OR

```
<ADD_CHILDREN DocVar="parentDoc" FromSelectList="X-PathExpr"/>
```

*Attributes:*

- DocVar: Parent document name.
- FromDocVar: Child document name.
- FromVar: Child document name (un-typed).
- FromSelect: Child document name.
- FromSelectList: List of child document names.
- FromValue: X-Path expression to select child document name(s). This is the recommended method to refer to child documents for performance reasons.

*Example 1:*

Assuming that the document variable 'myDoc' points to the following document instance:

```
<MY_DOCUMENT Comment="Printing my document">
<MY_CHILD1 Name="First Child" Value="1"/>
<MY_CHILD2 Name="Second Child" Value="2">
<MY_CHILD21 Name="Second-one Child" Value="2-1"/>
</MY_CHILD2>
<MY_CHILD3 Name="Third Child1" Value="30"/>
<MY_CHILD3 Name="Third Child2" Value="31"/>
</MY_DOCUMENT>
```

The following statement will add the first child of 'myDoc' with the name 'MY_CHILD2' to the DocVar 'thisReturn'.

```
<ADD_CHILDREN DocVar="thisReturn" FromValue="{$myDoc/MY_CHILD2}"/>
```

The DocVar can be printed to the log to check whether the operation was successful:

Input:

```
<PRINTLN Value="{$thisReturn}" />
```

Output:

```
<RESPONSE>
<MY_CHILD2 Name="Second Child" Value="2">
<MY_CHILD21 Name="Second-one Child" Value="2-1"/>
</MY_CHILD2>
</RESPONSE>
```

*Example 2:*

This is a variation of example 1. The following statement will add all children of 'myDoc' with the name `MY_CHILD3` to the variable '`thisReturn`'.

```
<ADD_CHILDREN DocVar="thisReturn" FromSelectList="$myDoc/MY_CHILD3"/
>
```

The DocVar can be printed to the log to check whether the operation was successful:

Input:

```
<PRINTLN Value="{$thisReturn}"/>
```

Output:

```
<RESPONSE>
<MY_CHILD3 Name="Third Child1" Value="30"/>
<MY_CHILD3 Name="Third Child2" Value="31"/>
</RESPONSE>
```

*Example 3:*

Input:

```
<TO_DOCVAR AssignToVar="parentDoc">
<DOC_ROOT>
<A_CHILD/>
<B_CHILD>
<EXISTING_B_CHILD/>
</B_CHILD>
</DOC_ROOT>
</TO_DOCVAR>
<TO_DOCVAR AssignToVar="childDoc">
<B_CHILD>
<NEW_B1_CHILD/>
<NEW_B2_CHILD/>
</B_CHILD>
</TO_DOCVAR>
<ADD_CHILDREN DocVar="parentDoc" FromDocVar="childDoc"/>
<PRINTLN Value="{$parentDoc}"/>
```

Output:

```
<DOC_ROOT>
```

```
<A_CHILD/>
<B_CHILD>
<EXISTING_B_CHILD/>
</B_CHILD>
<B_CHILD>
<NEW_B1_CHILD/>
<NEW_B2_CHILD/>
</B_CHILD>
</DOC_ROOT>
```

## APPEND_CHILDREN

*Description:*

The `APPEND_CHILDREN` statement is used to add one or more documents specified by the document variable '`childDoc`' to the document Variable 'parentDoc', it differs from the `ADD_CHILDREN` action in the fact that only the child tags from the document 'childDoc' are added without the enclosing parent tags.

*Syntax:*

```
<APPEND_CHILDREN DocVar="parentDoc" FromDocVar ="childDoc"
ChildName="ChildName"/>
```

*Attributes:*

- DocVar: Parent document name.
- FromDocVar: Child document name.
- ChildName: If specified, only the child elements with this name will be removed.

*Example 1:*

Input:

```
<TO_DOCVAR AssignToVar="parentDoc">
<DOC_ROOT>
<A_CHILD/>
<B_CHILD>
<EXISTING_B_CHILD/>
</B_CHILD>
</DOC_ROOT>
</TO_DOCVAR>
<TO_DOCVAR AssignToVar="childDoc">
```

```
<B_CHILD>
<NEW_B1_CHILD/>
<NEW_B2_CHILD/>
</B_CHILD>
</TO_DOCVAR>
<APPEND_CHILDREN DocVar="parentDoc" FromDocVar="childDoc"/>
<PRINTLN Value="{$parentDoc}"/>
```

Output:

```
<DOC_ROOT>
<A_CHILD/>
  <B_CHILD>
    <EXISTING_B_CHILD/>
  </B_CHILD>
  <NEW_B1_CHILD/>
  <NEW_B2_CHILD/>
</DOC_ROOT>
```

## REMOVE_CHILDREN

*Description:*

The `REMOVE_CHILDREN` statement is used to remove either all or a specific child element from the parent document, represented by `` `parentDoc' `` variable. Child elements of only one name can be removed at a time. If the AssignToVar attribute is specified, then the list of the removed child elements is assigned to the mentioned variable.

*Syntax:*

```
<REMOVE_CHILDREN DocVar="parentDoc"/>
```

OR

```
<REMOVE_CHILDREN ChildName ="ChildName" DocVar="parentDoc"/>
```

*Attributes:*

- DocVar: Parent document.
- ChildName: If specified, only the child elements with this name will be removed.
- AssignToVar: The list of removed child elements is assigned to this variable.

*Example 1:*

Assuming the DocVar `` `myDoc' `` points to the following document instance:

```
<MY_DOCUMENT Comment="This is my document">
```

```
<MY_CHILD1 Name="First Child" Value="1"/>
<MY_CHILD2 Name="Second Child" Value="2">
<MY_CHILD21 Name="Second-one Child" Value="2-1"/>
</MY_CHILD2 >
<MY_CHILD3 Name="Third Child1" Value="30"/>
<MY_CHILD3 Name="Third Child2" Value="31"/>
</MY_DOCUMENT>
```

The following statement will remove all the children of myDoc:

```
<REMOVE_CHILDREN DocVar="myDoc"/>
```

To verify, print the variable:

Input:

```
<PRINTLN Value="{$myDoc}"/>
```

Output:

```
<MY_DOCUMENT Comment="This is my document">
```

*Example 2:*

For the same document instance `myDoc' from Example 1, the following statement will remove all children with the name `MY_CHILD3':

```
<REMOVE_CHILDREN DocVar="myDoc" ChildName ="MY_CHILD3"/>
```

Input:

```
<PRINTLN Value="{$myDoc}" />
```

Output:

Printing my document:

```
<MY_DOCUMENT Comment="Printing my document">
<MY_CHILD1 Name="First Child" Value="1"/>
<MY_CHILD2 Name="Second Child" Value="2">
<MY_CHILD21 Name="Second-one Child" Value="2-1"/>
</MY_CHILD2>
</MY_DOCUMENT>
```

## REMOVE_NODES

*Description:*

The REMOVE_NODES statement is used to remove specific child element from the parent document.

*Syntax:*

```
<REMOVE_NODES FromSelectList="$origXML/UI_DYNAMIC_TABLE/table/field-
group" SelectList="$origXML/UI_DYNAMIC_TABLE/table/field-group/
display-field[@name='SYS_TARGET_ID']"/>
```

*Attributes:*

- FromSelectList: Parent document
- SelectList: Only the child elements with this name will be removed.

*Example 1:*

Assuming the DocVar `myDoc' points to the following document instance:

```
<DOCUMENTS>

<DEFINE_METHOD Name="testDelete1">

    <RULE>

      <ACTION>

        <TO_DOCVAR AssignToVar="origXML">

          <RESPONSE Status="Success">

            <UI_DYNAMIC_TABLE>

              <ui:table xmlns:ui="http://www.teradata.com/ui"
id="ERROR_TABLE" rememberSelections="false" rowId="ERROR_ID"
doPaging="true" displayText="error_HeadCount" rowSrc="$root/RECORDS/
*" scrollable="true" selectable="Single" postRowIdAs="SELECTED_ID"
displayTextNoRecords="No items found" totalRowCount="$root/
Report_Detail_TotalRowCount/@Value" startAtRow="$root/
Report_Detail_StartRowCount/@Value" maxRows="10">

                <ui:field-group xmlns:ui="http://www.teradata.com/u"
id="FieldGroup">

                  <ui:display-field name="AcctMonth" id="AcctMonth"
displayText="AcctMonth" dataType="string"/>

                  <ui:display-field name="AcctYear" id="AcctYear"
displayText="AcctYear" dataType="Number"/>

                  <ui:display-field name="Store" id="Store"
displayText="Store" dataType="Number"/>

                  <ui:display-field name="Type" id="Type"
displayText="Type" dataType="Number"/>

                  <ui:display-field name="CostCenterDepartment"
id="CostCenterDepartment" displayText="CostCenterDepartment"
dataType="string"/>
```

```
                    <ui:display-field name="Qty" id="Qty"
displayText="Qty" dataType="Number"/>

                    <ui:display-field name="SYS_TARGET_ID"
id="SYS_TARGET_ID" displayText="SYS_TARGET_ID" dataType="Number"/>

                    <ui:display-field name="SYS_AUTH_ID"
id="SYS_AUTH_ID" displayText="SYS_AUTH_ID" dataType="string"/>

                    <ui:display-field name="SOURCE" id="SOURCE"
displayText="SOURCE" dataType="string"/>

                    <ui:display-field name="ERROR_ID" id="ERROR_ID"
displayText="ERROR_ID" dataType="identity"/>

                    <ui:display-field name="ERROR_MESSAGE"
id="ERROR_MESSAGE" displayText="ERROR_MESSAGE" dataType="string"/>

                  <ui:display-field name="ERROR_TIME" id="ERROR_TIME"
displayText="ERROR_TIME" dataType="DateTime"/>

                    <ui:display-field name="SYS_ERR_SEVERITY"
id="SYS_ERR_SEVERITY" displayText="SYS_ERR_SEVERITY"
dataType="string"/>

                    <ui:display-field name="SYS_TYPE" id="SYS_TYPE"
displayText="SYS_TYPE" dataType="string"/>

                    <ui:display-field name="SYS_DOCUMENT"
id="SYS_DOCUMENT" displayText="SYS_DOCUMENT" dataType="string"/>

                    <ui:display-field name="SYS_LOAD_ID"
id="SYS_LOAD_ID" displayText="SYS_LOAD_ID" dataType="string"/>

                    <ui:display-field name="WORKFLOW_ID"
id="WORKFLOW_ID" displayText="WORKFLOW_ID" dataType="string"/>

                  </ui:field-group>

                  <ui:buttons id="buttons1" name="buttons1">

                  <ui:button displayText="Edit" id="Edit" name="Editl"
enabledBySelections="true"/>

                    <ui:button displayText="Reload" id="Reload"
name="Reload" enabledBySelections="true"/>

                  </ui:buttons>

                </ui:table>

              </UI_DYNAMIC_TABLE>

            </RESPONSE>

          </TO_DOCVAR>
```

```
        <REMOVE_NODES FromSelectList="$origXML/UI_DYNAMIC_TABLE/
table/field-group" SelectList="$origXML/UI_DYNAMIC_TABLE/table/
field-group/display-field[@name='SYS_TARGET_ID']"/>

        <SET DocVar="thisReturn" FromValue="{$origXML}"/>

      </ACTION>

    </RULE>

  </DEFINE_METHOD>

</DOCUMENTS>
```

The above code will remove the mentioned child element from the parent, and you will get the following output

```
<RESPONSES Status="Success">

          <RESPONSE Status="Success">

            <UI_DYNAMIC_TABLE>

              <ui:table xmlns:ui="http://www.teradata.com/ui"
id="ERROR_TABLE" rememberSelections="false" rowId="ERROR_ID"
doPaging="true" displayText="error_HeadCount" rowSrc="$root/RECORDS/
*" scrollable="true" selectable="Single" postRowIdAs="SELECTED_ID"
displayTextNoRecords="No items found" totalRowCount="$root/
Report_Detail_TotalRowCount/@Value" startAtRow="$root/
Report_Detail_StartRowCount/@Value" maxRows="10">

                <ui:field-group xmlns:ui="http://www.teradata.com/u"
id="FieldGroup">

                  <ui:display-field name="AcctMonth" id="AcctMonth"
displayText="AcctMonth" dataType="string"/>

                  <ui:display-field name="AcctYear" id="AcctYear"
displayText="AcctYear" dataType="Number"/>

                  <ui:display-field name="Store" id="Store"
displayText="Store" dataType="Number"/>

                  <ui:display-field name="Type" id="Type"
displayText="Type" dataType="Number"/>

                  <ui:display-field name="CostCenterDepartment"
id="CostCenterDepartment" displayText="CostCenterDepartment"
dataType="string"/>

                  <ui:display-field name="Qty" id="Qty"
displayText="Qty" dataType="Number"/>

                  <ui:display-field name="SYS_AUTH_ID"
id="SYS_AUTH_ID" displayText="SYS_AUTH_ID" dataType="string"/>

                  <ui:display-field name="SOURCE" id="SOURCE"
displayText="SOURCE" dataType="string"/>

                  <ui:display-field name="ERROR_ID" id="ERROR_ID"
displayText="ERROR_ID" dataType="identity"/>

                  <ui:display-field name="ERROR_MESSAGE"
id="ERROR_MESSAGE" displayText="ERROR_MESSAGE" dataType="string"/>
```

```
                    <ui:display-field name="ERROR_TIME" id="ERROR_TIME"
displayText="ERROR_TIME" dataType="DateTime"/>
                     <ui:display-field name="SYS_ERR_SEVERITY"
id="SYS_ERR_SEVERITY" displayText="SYS_ERR_SEVERITY"
dataType="string"/>
                     <ui:display-field name="SYS_TYPE" id="SYS_TYPE"
displayText="SYS_TYPE" dataType="string"/>
                     <ui:display-field name="SYS_DOCUMENT"
id="SYS_DOCUMENT" displayText="SYS_DOCUMENT" dataType="string"/>
                     <ui:display-field name="SYS_LOAD_ID"
id="SYS_LOAD_ID" displayText="SYS_LOAD_ID" dataType="string"/>
                     <ui:display-field name="WORKFLOW_ID"
id="WORKFLOW_ID" displayText="WORKFLOW_ID" dataType="string"/>
                   </ui:field-group>
                   <ui:buttons id="buttons1" name="buttons1">
                   <ui:button displayText="Edit" id="Edit" name="Editl"
enabledBySelections="true"/>
                     <ui:button displayText="Reload" id="Reload"
name="Reload" enabledBySelections="true"/>
                   </ui:buttons>
                 </ui:table>
               </UI_DYNAMIC_TABLE>
             </RESPONSE>
           </RESPONSES>
```

## DELETE_ALL_CHILDREN

*Description:*

This statement will remove a specified list of different child elements from the parent XML. It should be used in preference to `REMOVE_CHILDREN` when children with different names must be removed from the same XML. If the `AssignToVar` variable is specified, all the removed children are returned in a List structure.

*Syntax:*

```
<DELETE_ALL_CHILDREN SelectList="$varList" {AssignToVar="delNodes"}>

</DELETE_ALL_CHILDREN>
```

*Attributes:*

- Select: Parent document.
- SelectList: List of parent documents.
- AssignToVar: Variable which gets assigned all the removed child elements.

*Example 1:*

If you give the following input:

```
<TO_DOCVAR AssignToVar="myXML">
<COUNTRY Value="India">
<STATE Value="Maharashtra">
<CAPITAL Value="Bombay"/>
</STATE>
<STATE Value="Karnataka">
<CAPITAL Value="Bangalore"/>
</STATE>
<POPULATION Value="80C"/>
</COUNTRY>
</TO_DOCVAR>
<DELETE_ALL_CHILDREN Select="$myXML">
<CAPITAL/>
<POPULATION/>
</DELETE_ALL_CHILDREN>
<PRINTLN Value="{$myXML}"/>
```

The output would be:

```
<COUNTRY Value="India">
<STATE Value="Maharashtra"/>
<STATE Value="Karnataka"/>
</COUNTRY>
```

## COLLATE_XML

*Description:*

This statement collates the input XML according to the collate list specified by the `COLLATE_LIST` tags in the input XML.

*Syntax:*

```
<COLLATE_XML Var="docVariable" AssignToVar="res"/>
```

*Syntax:*

```
<MY_DOCUMENT {Comment="..."}>
<INPUT>
<TAG_A Value="A1"/>
<TAG_A Value="A2"/>
```

```
.....................
<TAG_A Value="An"/>
<TAG_B Value="B1"/>
<TAG_B Value="B2"/>
.....................
<TAG_B Value="Bn"/>
.....................
.....................
<TAG_X Value="X1"/>
<TAG_X Value="X2"/>
.....................
<TAG_X Value="Xn"/>
</INPUT>
<COLLATE_LIST Value="OUTPUT">
<TAG_A/>
<TAG_B/>
............
<TAG_X/>
</COLLATE_LIST>
</MY_DOCUMENT>
```

*Syntax:*

```
<OUTPUT_LISTS>
<OUTPUT>
<TAG_A Value="A1"/>
<TAG_B Value="B1"/>
............
<TAG_X Value="X1"/>
</OUTPUT>
<OUTPUT>
<TAG_A Value="A2"/>
<TAG_B Value="B2"/>
............
```

```
<TAG_X Value="X2"/>
</OUTPUT>
........
........
<OUTPUT>
<TAG_A Value="An"/>
<TAG_B Value="Bn"/>
............
<TAG_X Value="Xn"/>
</OUTPUT>
</OUTPUT_LISTS>
```

*Attributes:*

- Var: Document variable that will be collated.
- AssignToVar: Document variable that contains the output of the collation.

*Example 1:*

Assuming that myDoc points to the following document instance:

```
<MY_DOCUMENT Comment="Printing my document">
<INPUT>
<AO_ID Value="PO-128"/>
<AO_ID Value="PO-127"/>
<AO_ID Value="PO-126"/>
<AO_ID Value="PO-125"/>
<ITEM_ID Value="136D8710G01-B"/>
<ITEM_ID Value="136D8710G01-B"/>
<ITEM_ID Value="136D8710G01-B"/>
<ITEM_ID Value="136D8710G01-B"/>
<ID Value="SCH-128"/>
<ID Value="SCH-127"/>
<ID Value="SCH-126"/>
<ID Value="SCH-125"/>
</INPUT>
<COLLATE_LIST Value="LI_DETAILS">
<ID/>
```

```
<AO_ID/>
<ITEM_ID/>
</COLLATE_LIST>
</MY_DOCUMENT>
```

The following statements will execute the collate statement print the results:

```
<COLLATE_XML DocVar="myDoc" AssignToVar="collateDoc"/>
<PRINTLN Value="{$collateDoc }" />
```

Output on the console:

```
<LI_DETAILS_LIST>
<LI_DETAILS>
<ITEM_ID Value="136D8710G01-B"/>
<AO_ID Value="PO-128"/>
<ID Value="SCH-128"/>
</LI_DETAILS>
<LI_DETAILS>
<ITEM_ID Value="136D8710G01-B"/>
<AO_ID Value="PO-127"/>
<ID Value="SCH-127"/>
</LI_DETAILS>
<LI_DETAILS>
<ITEM_ID Value="136D8710G01-B"/>
<AO_ID Value="PO-126"/>
<ID Value="SCH-126"/>
</LI_DETAILS>
<LI_DETAILS>
<ITEM_ID Value="136D8710G01-B"/>
<AO_ID Value="PO-125"/>
<ID Value="SCH-125"/>
</LI_DETAILS>
</LI_DETAILS_LIST>
```

## MAP

*Description:*

Map statements can be used to group elements based on certain properties that becomes the key. There are statements to create an empty map, populate map, clear map, and check for elements in the map and to remove elements from a map.

*Syntax:*

To create an empty map:

```
<MAP_CREATE AssignToVar="myMap"/>
```

To create a map with elements populated:

```
<MAP_CREATE SelectList="$data/CARS/CAR" CurrentElement="e"
Key="key($e/MAKE/@Value, $e/YEAR/@Value)" AssignToVar="carMap"/>
```

To add elements to a map:

```
<MAP_PUT_ALL Map="$carMap" SelectList="$newData/*"
CurrentElement="e" Key="key($e/MAKE_YEAR/@Value)"/>
```

*Attributes:*

None

*Examples:*

To access a map element:

```
<SET Var="myCar" FromSelect="mapGet($carMap, key('Nissan','2003'))"/
>
```

To get a map key:

```
<SET Var="key1" FromSelect="$carMap/KEY/@Value"/>
```

To get a map key's part:

```
<SET Var="key1Part1" FromSelect="$carMap/KEY[1]/PART[1]/@Value"/>
```

To check the presence of element:

```
<SET Var="containsKeyop1" FromSelect="mapContainsKey($carMap,
key('Nisan','2001'))"/>
```

To access key and values in a loop:

```
<FOR_EACH SelectList="$carMap/KEY" CurrentElement="currentKey">

<PRINTLN Var="currentKey"/>

<SET Var="keyValueElements" FromSelect="mapGet($carMap,
key($currentKey/@Value))"/>

<FOR_EACH SelectList="$keyValueElements/*"
CurrentElement="keyValueElement">

<PRINTLN Var="keyValueElement"/>

</FOR_EACH>

</FOR_EACH>
```

To remove a map element:

```
<MAP_REMOVE Map="$carMap" Key="key('Honda','2002')"/>
```

To clear all map elements:

```
<MAP_CLEAR Map="$carMap"/>
```

## GROUP_DOCS

*Description:*

This action will group a given list of document instances based on the specified criteria. This action tag can be used wherever grouping is required.

If the list size is not too large, using this tag will give better performance over using MAP action tags.

*Syntax:*

```
<GROUP_DOCS SelectList="list" AssignToVar="myVar">
<PROPERTIES>
<Property1/>
< Property2/>
.........
</PROPERTIES>
<GROUP_NAME Value="groupName"/>
<DOCS_NAME Value="docName"/>
<ROOT_NAME Value="rootName"/>
</GROUP_DOCS>
```

*Attributes:*

- SelectList (Required): List of XML documents for grouping
- AssignToVar (Required): Grouping result is assigned to this variable

*Example 1:*

```
<GROUP_DOCS SelectList="$thisParam/ORDER_SCHEDULE"
AssignToVar="thisReturn">
<PROPERTIES>
<SHIP_FROM_NAME/>
<SHIP_TO_NAME/>
</PROPERTIES>
<GROUP_NAME Value="SHIPMENT"/>
<DOCS_NAME Value="ORDER_SCHEDULES"/>
```

```
<ROOT_NAME Value="RESPONSE"/>
</GROUP_DOCS>
```

If the request is like the following, after applying `GROUP_DOCS`, the response will be as mentioned under the `RESPONSE`:

```
<REQUEST Name="groupDocs">
<ORDER_SCHEDULE>
<ID Value="SCH-1"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
<ORDER_SCHEDULE>
<ID Value="SCH-2"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-Atl"/>
</ORDER_SCHEDULE>
<ORDER_SCHEDULE>
<ID Value="SCH-3"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
<ORDER_SCHEDULE>
<ID Value="SCH-4"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
</REQUEST>
```

Response after grouping:

```
<RESPONSES Status="Success">
<RESPONSE Status="Success">
<SHIPMENT>
<SHIP_FROM_NAME Value="teradata-Atl"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<ORDER_SCHEDULES>
```

```
<ORDER_SCHEDULE>
<ID Value="SCH-2"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-Atl"/>
</ORDER_SCHEDULE>
</ORDER_SCHEDULES>
</SHIPMENT>
<SHIPMENT>
<SHIP_FROM_NAME Value="teradata-MtView"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<ORDER_SCHEDULES>
<ORDER_SCHEDULE>
<ID Value="SCH-1"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
<ORDER_SCHEDULE>
<ID Value="SCH-3"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
<ORDER_SCHEDULE>
<ID Value="SCH-4"/>
<SHIP_TO_NAME Value="teradata-Dallas"/>
<SHIP_FROM_NAME Value="teradata-MtView"/>
</ORDER_SCHEDULE>
</ORDER_SCHEDULES>
</SHIPMENT>
</RESPONSE>
</RESPONSES>
```

## MERGE_DOC

*Description:*

The MERGE_DOC operation will merge 2 document instances, one of which is considered the new one and the other the older version. The new instance is referred to in the Lhs expression and the older version in the Rhs expression.

The tag name needs to correspond to a Document defined in the service. If the tag name does not correspond to a Document, a DocName attribute needs to be specified to indicate what Document metadata to use.

*Syntax:*

```
<MERGE_DOC Value="newDoc" FromValue="oldDoc"/>
```

*Attributes:*

- Value (Required): New document
- FromValue (Required): Old document
- DocName (Optional): Used when tag names are different

*Example 1:*

```
<MERGE_DOC Value="{$thisParam/ORDER_SCHEDULE}"
FromValue="{$thisParam/DOC/ORDER_SCHEDULE}"/>
```

The variable thisParam refers to the following document:

```
<PARENT_DOC>
<!-- This ORDER_SCHEDULE instance is considered to be the new one -->
<ORDER_SCHEDULE>
<ID Value="SCHED-1"/>
<QTY Value="10"/>
<DAYS_LATE Value="10"/>
<DELIVERY_DATE Value="10/20/2005 00:10:20"/>
<REASON_CODE Value="NB"/>
<CITY_TAX/>
<_RESULT Value="SUCCESS"/>
</ORDER_SCHEDULE>
<DOC>
<!-- This ORDER_SCHEDULE instance is considered to be the old one -->
<ORDER_SCHEDULE Value="2">
<ID Value="SCH-101"/>
<REASON_CODE Value="NB"/>
<QTY Value="900.0"/>
<CITY_TAX Value="92.4"/>
```

```
<VERSION_NO Value="72"/>
<DAYS_EARLY Value="1"/>
<DELIVERY_DATE Value="09/20/2001 00:10:20"/>
</ORDER_SCHEDULE>
</DOC>
</PARENT_DOC>
```

This is the response after the `MERGE_DOC` operation on the request XML:

```
<RESPONSE>
<ORDER_SCHEDULE>
<ID Value="SCHED-1" ChangeType="Change" OldValue="SCH-101"/>
<QTY Value="10" ChangeType="Change" OldValue="900.0"/>
<DAYS_LATE Value="10"/>
<DELIVERY_DATE Value="10/20/2005 00:10:20" ChangeType="Change"
OldValue="09/20/2001 00:10:20:000"/>
<REASON_CODE Value="NB" ChangeType="None" OldValue="NB"/>
<CITY_TAX ChangeType="Change" OldValue="92.4"/>
<_RESULT Value="SUCCESS"/>
<VERSION_NO Value="72" ChangeType="None" OldValue="72"/>
</ORDER_SCHEDULE>
</RESPONSE>
```

## MERGE_ERRORS_DOC

*Description:*

.

*Syntax:*

```
<MERGE_ERRORS_DOC .../>
```

*Attributes:*

*Example:*

## STRING_TO_XML

*Description:*

This action tag will convert a string representing XML as read from a file into an XML form instance.

*Syntax:*

```
<STRING_TO_XML FromSelect="$xmlString" DocVar="xmlFormInstance"/>
```

*Attributes:*

- DocVar (Required): Variable to assign the XML form object to
- FromValue (Optional): string variable representing XML
- FromSelect (Optional): string variable representing XML
- FromVar (Optional): string variable representing XML
- FromDocVar (Optional): string variable representing XML
- FromSelectList (Optional): string variable representing XML

Exactly one of the RhsExpression - FromValue, FromVar, FromSelect, FromSelectList, FromDocVar - should be specified.

*Example 1:*

```
<STRING_TO_XML FromSelect="$xmlString" DocVar="xmlFormInstance"/>
```

## XML_TO_STRING

*Description:*

This action tag will convert an XML form instance into a string.

*Syntax:*

```
<XML_TO_STRING FromSelect="$xml" DocVar="xmlString"/>
```

*Attributes:*

- DocVar (Required): Variable to assign the XML form object to
- FromValue (Optional): string variable representing XML
- FromSelect (Optional): string variable representing XML
- FromVar (Optional): string variable representing XML
- FromDocVar (Optional): string variable representing XML
- FromSelectList (Optional): string variable representing XML

Exactly one of the RhsExpression - FromValue, FromVar, FromSelect, FromSelectList, FromDocVar - should be specified.

*Example 1:*

```
<XML_TO_STRING FromSelect="$xml" Var="xmlString"/>
```

## CLONE_XML

*Description:*

The `CLONE_XML` operation returns a clone of the XML form instance.

*Syntax:*

```
<CLONE_XML DocVar="XMLformObject" AssignToVar="myVar"/>
```

*Attributes:*

- DocVar (Required): XML form object to clone
- AssignToVar (Required): Variable to assign the cloned XML to

*Example 1:*

Input:

```
<TO_DOCVAR AssignToVar="myXML">
<COUNTRY Value="India">
<STATE Value="Maharashtra">
<CAPITAL Value="Bombay"/>
</STATE>
<POPULATION Value="80C"/>
</COUNTRY>
</TO_DOCVAR>
<CLONE_XML DocVar="myXML" AssignToVar="clonedXML"/>
<PRINTLN Value="{$clonedXML}"/>
```

Output:

```
<COUNTRY Value="India">
<STATE Value="Maharashtra">
<CAPITAL Value="Bombay"/>
</STATE>
<POPULATION Value="80C"/>
</COUNTRY>
```

## SORT_DOCS

*Description:*

The `SORT_DOCS` action tag will sort a given list of document instances based on the specified criteria. This action tag can be used wherever sorting is required.

*Syntax:*

```
<SORT_DOCS SelectList="list" AssignToVar="myVar">
<Property1 Type="xcore-data-type" Sort="Ascending|Descending"
EnumList="comma-seperated-list for enum"/>
<Property2 Type=="xcore-data-type" Sort="Ascending|Descending"
EnumList="comma-seperated-list for enum"/>
```

```
 .........
</SORT_DOCS>
```

*Attributes:*

- SelectList (Required): List of XML documents for sorting.
- AssignToVar (Required): Sorted result is assigned to this variable.
- Type (Optional): Should be a valid xcore data type or enum. The xcore data types are int, string, date, double, float, timestamp, datetime, and boolean.
- EnumList (Optional): This is a comma separated list of enum values. This is needed if `Type=enum`

*Example 1:*

```
<DEFINE_METHOD Name="sortTest">
    <RULE>
      <ACTION>
        <SORT_DOCS SelectList="$thisParam/A"
        AssignToVar="myList">
          <STATE Sort="Descending" Type="enum"
          EnumList="B,D,C,A,E"/>
          <PROP1 Sort="Ascending" Type="int"/>
          <PROP3 Sort="Ascending" Type="date"/>
        </SORT_DOCS>
        <APPEND_TO_XML DocVar="thisReturn">
          <TO_XML SelectList="$myList"/>
        </APPEND_TO_XML>
      </ACTION>
     </RULE>
   </DEFINE_METHOD>
```

If the request is like the following, after applying `SORT_DOCS`, the response will be as mentioned under the `RESPONSE`:

```
<REQUEST Name="sortTest">
        <A>
            <PROP1 Value="10" />
            <PROP2 Value="M" />
            <PROP3 Value="01/01/2010" />
            <STATE Value="A"/>
```

```
</A>
<A>
    <PROP1 Value="5" />
    <PROP2 Value="B" />
    <PROP3 Value="01/01/2007" />
    <STATE Value="B"/>
</A>
<A>
    <PROP1 Value="7" />
    <PROP2 Value="M" />
    <PROP3 Value="01/01/2010" />
    <STATE Value="B"/>
</A>
<A>
    <PROP1 Value="7" />
    <PROP2 Value="I" />
    <PROP3 Value="01/01/2003" />
    <STATE Value="C"/>
</A>
<A>
    <PROP1 Value="1" />
    <PROP2 Value="C" />
    <PROP3 Value="01/01/2005" />
    <STATE Value="A"/>
</A>
<A>
    <PROP1 Value="7" />
    <PROP2 Value="X" />
    <PROP3 Value="01/01/2001" />
    <STATE Value="D"/>
</A>
<A>
```

```
            <PROP1 Value="5" />
            <PROP2 Value="B" />
            <PROP3 Value="01/01/2015" />
            <STATE Value="E"/>
        </A>
    </REQUEST>
```

Response after sorting:

```
<RESPONSE Status="Success">
                <A>
                  <PROP1 Value="5"/>
                  <PROP2 Value="B"/>
                  <PROP3 Value="01/01/2015"/>
                  <STATE Value="E"/>
                </A>
                <A>
                  <PROP1 Value="1"/>
                  <PROP2 Value="C"/>
                  <PROP3 Value="01/01/2005"/>
                  <STATE Value="A"/>
                </A>
                <A>
                  <PROP1 Value="10"/>
                  <PROP2 Value="M"/>
                  <PROP3 Value="01/01/2010"/>
                  <STATE Value="A"/>
                </A>
                <A>
                  <PROP1 Value="7"/>
                  <PROP2 Value="I"/>
                  <PROP3 Value="01/01/2003"/>
                  <STATE Value="C"/>
                </A>
```

```
    <A>
      <PROP1 Value="7"/>
      <PROP2 Value="X"/>
      <PROP3 Value="01/01/2001"/>
      <STATE Value="D"/>
    </A>
    <A>
      <PROP1 Value="5"/>
      <PROP2 Value="B"/>
      <PROP3 Value="01/01/2007"/>
      <STATE Value="B"/>
    </A>
    <A>
      <PROP1 Value="7"/>
      <PROP2 Value="M"/>
      <PROP3 Value="01/01/2010"/>
      <STATE Value="B"/>
    </A>
  </RESPONSE>
```

## SORT

*Description:*

The `SORT` action tag does an in place sort of a list of XMLform instances based on the specified property.

*Syntax:*

```
<SORT SelectList="XMLformList" Key="Property" DataType="Type"
SortOrder="ascending/descending"/>
```

*Attributes:*

- SelectList (Required): XML form list
- Key (Required): Property to sort the list on
- DataType (Optional): Type of data being sorted; string by default.
- SortOrder (Optional): Ascending/descending; ascending by default.

*Example 1:*

This is the list to be sorted in descending order using `NAME` property as the key:

```
<PARAMS>
<COUNTRY>
<NAME Value="France"/>
<CAPITAL Value="Paris"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Spain"/>
<CAPITAL Value="Madrid"/>
</COUNTRY>
<COUNTRY>
<NAME Value="UK"/>
<CAPITAL Value="London"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Germany"/>
<CAPITAL Value="Berlin"/>
</COUNTRY>
</PARAMS>
```

This is the input command:

```
<SORT SelectList="$myXml/COUNTRY" DataType="string" Key="NAME"
SortOrder="descending"/>
```

This is the output list:

```
<COUNTRY>
<NAME Value="UK"/>
<CAPITAL Value="London"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Spain"/>
  <CAPITAL Value="Madrid"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Germany"/>
<CAPITAL Value="Berlin"/>
```

```
</COUNTRY>
<COUNTRY>
<NAME Value="France"/>
<CAPITAL Value="Paris"/>
</COUNTRY>
```

# Logical Operations

## IF_TEST, THEN, ELSE

*Description:*

The `IF_TEST-THEN-ELSE` statements form an execution-control-block in MDM and are used to test whether a condition, expressed as a Boolean X-Path expression, evaluates to true or false. If the condition evaluates to "true", then the action statements associated with `THEN` is executed. Otherwise, the action statements associated with the "`ELSE`" statement are executed.

Use of X-Path expressions more than 80 characters should be avoided for readability. IF_TEST is generally preferred over IF statement whenever the test condition can be expressed as a compact expression.

*Syntax:*

```
<IF_TEST Test="X-Path Expression that evaluates to Boolean">
<THEN>
...
</THEN>
<ELSE>
...
</ELSE>
</IF_TEST>
```

*Attributes:*

Test (Required): The test condition expressed as a Boolean X-Path Expression

*Example:*

Assuming that '`coDoc`' points to the following document instance:

```
<CUSTOMER_ORDER>
<ID Value="CO-01"/>
<ORDER_TYPE_ID Value="STANDARD"/>
```

```
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-01"/>
<ITEM_ID Value="CPU"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-02"/>
<ITEM_ID Value="MONITOR"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

*Example 1:*

IF_TEST Condition that Evaluates to True

Input:

```
<IF_TEST Test="$coDoc/ID/@Value = 'CO-01'">
<THEN>
<PRINT Value="Found CO"/>
</THEN>
</IF_TEST>
```

Output:

```
Found CO
```

*Example 2:*

IF_TEST Condition that Evaluates to False

Input:

```
<IF_TEST Test="$coDoc/CUSTOMER_TYPE = null">
<THEN>
<PRINT Value="Exception. Missing CUSTOMER_TYPE in CUSTOMER_ORDER."/>
<EXIT/>
</THEN>
</IF_TEST>
```

Output:

```
Exception. Missing CUSTOMER_TYPE in CUSTOMER_ORDER.
```

## IF, THEN, ELSE

*Description:*

The `IF-THEN-ELSE` statements form an execution-control-block in MDM and are used to test a condition and perform actions based on the output of the condition. The condition statement can be a simple condition or a compound condition. If there is more than one condition statement, the statements are combined into a single compound condition statement using "`AND`" and "`OR`" operators. If the condition evaluates to "true", then the action statements associated with `THEN` is executed. Otherwise, the action statements associated with the "`ELSE`" statement are executed.

The use of IF_TEST is generally preferred to that of IF whenever the Test condition can be expressed as a compact expression. But if the X-Path condition is complex, then writing it out as IF will make it more comprehensible.

*Syntax:*

```
<IF>
<COND>
{condition statement1}
.................
{condition statement N}
</COND>
<THEN>
{action statement1}
.................
{action statement N}
</THEN>
<ELSE> <!--ELSE is optional-->
{action statement1}
.................
{action statement N}
</ELSE>
</IF>
```

*Attributes:*

None

*Example:*

Input:

```
<IF>
```

```
<COND>

<IS_TRUE Value="{$lineItemDoc/TOTAL_PROMISED_QTY/
@Value=$lineItemDoc/TOTAL_REQUIRED_QTY/@Value}"/>

<IS_TRUE Value="{$lineItemDoc/STATUS/@Value != 'CANCEL'}"/>

<IS_TRUE Value="{isValidValue($lineItemDoc/TYPE/@Value, 'STANDARD',
'SAMPLE','TRANSFER','CONSIGNMENT')}"/>

</COND>

<THEN>

<PRINTLN Value="The total promised qty == required quantity"/>

</THEN>

<ELSE>

<PRINTLN Value="The condition failed"/>

</ELSE>

</IF>
```

Output (assuming the condition is true):

```
The total promised qty == required qty
```

*Example 2:*

The previous example written out as an `IF_TEST` looks more complex than writing it out as IF, and hence should be avoided.

Input:

```
<IF_TEST Test="(($lineItemDoc/TOTAL_PROMISED_QTY/
@Value=$lineItemDoc/TOTAL_REQUIRED_QTY/@Value) and ($lineItemDoc/
STATUS/@Value != 'CANCEL')) and isValidValue($lineItemDoc/TYPE/
@Value, 'STANDARD', 'SAMPLE','TRANSFER', 'CONSIGNMENT')">

<THEN>

<PRINTLN Value="The total promised qty == required quantity"/>

</THEN>

<ELSE>

<PRINTLN Value="The condition failed"/>

</ELSE>

</IF_TEST>
```

Output:

```
The total promised qty == required qty
```

## FOR_EACH, BREAK, CONTINUE

*Description:*

The FOR_EACH-BREAK-CONTINUE statements form an execution-control-block in MDM and are used to perform iterations. The action statements are executed for each element of the list SelectList, and the current element is referred by the 'CurrentElement' attribute.

CONTINUE statement will skip executing any following action statements for the current element and transfer the execution to the first statement after the start of the FOR_EACH tag.

BREAK statement will skip executing any following action statements for the current element and transfer the execution to the first statement after the end of the FOR_EACH tag.

*Syntax:*

```
<FOR_EACH SelectList="X-PathExpr" CurrentElement="elemVar">
{action statement1 }
.................
{action statement N }
</FOR_EACH>
```

*Attributes:*

- SelectList (Required): List of elements to perform actions on
- CurrentElement (Required): List element under inspection

*Example:*

First, a list of Purchase Order docs is assigned to the variable 'poDocs'. Then a FOR_EACH block iterates through these docs and count all the docs in 'COMPLETED' state. Finally, the number of docs in 'COMPLETED' state is printed out.

```
<GET_DOCUMENT Name="PURCHASE_ORDER" AssignToVar="poDocs">
<BUYER_ID Value="CISCO"/>
</GET_DOCUMENT>
<FOR_EACH SelectList="$poDocs/PURCHASE_ORDER"
CurrentElement="poDocVar">
<IF_TEST Test="$poDocVar/STATUS/@Value='COMPLETED'">
<THEN>
<SET Var="poCount" FromValue="{int(poCount)+1}"/>
</THEN>
<ELSE>
<CONTINUE/>
</ELSE>
</IF_TEST>
```

```
</FOR_EACH>

<PRINTLN Value="#POs in COMPLETED state:{$poCount}"/>
```

Output:

```
POs in COMPLETED state: 43
```

## REPEAT

*Description:*

The `REPEAT` statement is used within an X-Rule to perform a set of action statements repeatedly. The 'Count' attribute specifies the number of times the actions are repeated and the current index is referred by the 'IndexVar' attribute.

*Syntax:*

```
<REPEAT Count="X-PathExpr" IndexVar="currentIndex">

{action statement1}

.............

{action statement N}

</REPEAT>
```

*Attributes:*

- Count (Required): Number of times to repeat the actions
- IndexVar (Required): The current index value

*Example:*

An X-Rule 'createOrderXml' builds a customer order with a variable number of lines.

Method:

```
<DEFINE_METHOD Name="createOrderXml">

<RULE>

<ACTION>

<TO_DOCVAR AssignToVar="thisReturn">

<RESPONSE>

<CUSTOMER_ORDER>

<ID Value="2"/>

<REPEAT Count="int($thisParam/NUM_LINES/@Value)"
IndexVar="currIndex">

<SET Var="liId" FromValue="{concat('LI_', $currIndex)}"/>

<TO_DOCVAR AssignToVar="liDoc">

<ORDER_LINE_ITEM>
```

```
<ID Value="{string($liId)}"/>
</ORDER_LINE_ITEM>
</TO_DOCVAR>
<TO_XML DocVar="liDoc"/>
</REPEAT>
</CUSTOMER_ORDER>
</RESPONSE>
</TO_DOCVAR>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

Input:

```
<REQUEST Name="createOrderXml">
<NUM_LINES Value="5"/>
</REQUEST>
```

Output:

```
<RESPONSE>
<CUSTOMER_ORDER>
<ID Value="2"/>
<ORDER_LINE_ITEM>
<ID Value="LI_0"/>
</ORDER_LINE_ITEM>
..............
..............
<ORDER_LINE_ITEM>
<ID Value="LI_4"/>
</ORDER_LINE_ITEM>
</CUSTOMER_ORDER>
</RESPONSE>
```

## WHILE

*Description:*

The `WHILE` statement is used to perform a set of actions repeatedly as long as the specified condition is true.

*Syntax:*

```
<WHILE Condition="$condition=true()">
{action statement1}
..............
{action statement N}
</WHILE>
```

*Attributes:*

Condition (Required): Condition to evaluate

*Example:*

Input:

```
<SET Var="index" FromValue="1"/>
<TO_DOCVAR AssignToVar="thisReturn">
<RESPONSE>
<CUSTOMER_ORDER>
              <ID Value="CO_1"/>
              <WHILE Condition="$index &lt;=3">
<SET Var="liId" FromValue="{concat('LI_', $index)}"/>
<TO_DOCVAR AssignToVar="liDoc">
<ORDER_LINE_ITEM>
<ID Value="{string($liId)}"/>
</ORDER_LINE_ITEM>
</TO_DOCVAR>
<SET Var="index" FromValue="{int($index + 1)}"/>
                <TO_XML DocVar="liDoc"/>
              </WHILE>
</CUSTOMER_ORDER>
</RESPONSE>
</TO_DOCVAR>
```

Output:

```
<RESPONSE Status="Success">
<CUSTOMER_ORDER>
```

```
<ID Value="CO_1"/>
<ORDER_LINE_ITEM>
<ID Value="LI_1"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI_2"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI_3"/>
</ORDER_LINE_ITEM>
</CUSTOMER_ORDER>
</RESPONSE>
```

## EXIT

*Description:*

This statement is used to exit the X-Rule module successfully.

*Syntax:*

```
<EXIT/>
```

*Attributes:*

None

*Example:*

```
<IF_TEST Test="$thisParam/ID/@Value=null">
<THEN>
<EXIT/>
</THEN>
<ELSE>
<PRINTLN Value="This is a valid profile"/>
</ELSE>
</IF_TEST>
```

# User Interface Related Operations

## PAD

*Description:*

This tag is used to build the outermost (or parent) pad item in the navigation pane in the MDM UI. The UI framework invokes an X-Rule to get the list of navigation pad items; a `TO_DOCVAR` operation is used to build this list where PAD is used as the parent tag. This list is appended to the 'thisParam' variable and returned by the X-Rule to the UI framework.

*Syntax:*

```
<TO_DOCVAR AssignToVar="navigationXml">
<PAD Name="Studio" DisplayText="From Studio" Scrollable="yes">
<PAD_ITEM ...../>
.........
</PAD>
</TO_DOCVAR>
<ADD_CHILDREN DocVar="thisReturn" FromSelect="$navigationXml"/>
```

*Attributes:*

- Name (Required): Pad name (internal reference)
- DisplayText (Required): Text to display on the UI
- Scrollable (Optional): 'yes/no'; if set yes, then you can scroll the navigation pane

*Example 1:*

```
<TO_DOCVAR AssignToVar="navigationXml">
<PAD Name="Studio" DisplayText="From Studio" Scrollable="yes">
<PAD_ITEM OnClick="/base/start.x2ps?START_WORKFLOW=NPR"
DisplayText="NPR Workflow"/>
</PAD>
</TO_DOCVAR>
<ADD_CHILDREN DocVar="thisReturn" FromSelect="$navigationXml"/>
```

The left navigation pane will show:

```
From Studio
NPR Workflow
```

## PAD_ITEM

*Description:*

This tag is used as the child tag of `PAD` to represent a single navigation item in the left pane in the MDM UI. It contains an 'OnClick' attribute which contains the URL of the workflow (you can specify which service the workflow belongs to and if it should be opened in a separate window or the same); clicking on the URL will execute the workflow.

*Syntax:*

```
<TO_DOCVAR AssignToVar="navigationXml">

<PAD .......>

<PAD_ITEM OnClick="/base/
start.x2ps?START_WORKFLOW=yourworkflow&amp;SERVICE_NAME=yourservice"
DisplayText="text" Popup="true/false"/>

</PAD>

</TO_DOCVAR>

<ADD_CHILDREN DocVar="thisReturn" FromSelect="$navigationXml"/>
```

*Attributes:*

- OnClick (Required): URL to go-to when the navigation item is clicked; you have to specify the workflow name and the service name. If no service name is given, then the owner service is considered as default.
- DisplayText (Required): Text to display on the UI.
- Popup (Optional): 'true/false'; if set yes, then the URL will open in a new window (default is false).

*Example 1:*

```
<TO_DOCVAR AssignToVar="navigationXml">

<PAD Name="Studio" DisplayText="From Studio" Scrollable="yes">

<PAD_ITEM OnClick="/base/
start.x2ps?START_WORKFLOW=NewPartIntroduction&amp;SERVICE_NAME=ItemM
aster" Popup="true"/>

</PAD>

</TO_DOCVAR>

<ADD_CHILDREN DocVar="thisReturn" FromSelect="$navigationXml"/>
```

The left navigation pane will show:

```
From Studio

NPR Workflow
```

Clicking on `NPR Workflow` will open another window and launch the workflow.

## SET_SESSION

*Description:*

This tag is used to set session variables. The session is created when the user is logged in and it expires when the user session times out or when he/she log outs.

*Syntax:*

```
<SET_SESSION Name="varName" FromValue="value"/>
```

*Attributes:*

- Name (Required): Session variable name.
- FromValue (Required): Value to store in session variable.

*Example 1:*

```
<SET_SESSION Name="mySessionVar" FromValue="3"/>
```

## GET_SESSION

*Description:*

This tag is used to retrieve the session variables. The session is created when the user is logged in and it expires when the user session times out or when he/she log outs.

*Syntax:*

```
<GET_SESSION Name="value" AssignToVar="var"/>
```

*Attributes:*

- Name (Required): Session variable name.
- AssignToVar (Required): Variable to assign value from the session variable.
- HandleException (Optional): 'true/false'. Set 'true' to catch any exception raised by MDM.

*Example 1:*

```
<GET_SESSION Name="mySessionVar" AssignToVar="returnVar"
HandleException="true"/>
```

## FORM_SEARCH_FILTER

*Description:*

The `FORM_SEARCH_FILTER` tag is primarily used in the UI workflows to help formulate the search filters to query documents in the database based on the parameters specified in the PGL screen. The input to this action tag is the XML data returned from a search PGL screen (outXML). The criteria to formulate the filters are described as attributes of child elements.

*Syntax:*

```
<FORM_SEARCH_FILTER Value="{val}" AssignToVar="{varName}">

<ChildTag1/>

< ChildTag2 MultiValued="true"/>

< ChildTag3 Range="true"/>

< ChildTag4 Operator="true"/>
```

```
< ChildTag5 Value="{value}" AddOn="true"/>
</FORM_SEARCH_FILTER>
```

*Attributes:*

- Value: X-Path expression to specify the XML output from the PGL
- AssignToVar: Variable to assign the filter expression to

## Child Tag Attributes

- MultiValued: "true" indicates that the search filter for the tag must be formulated with all the values present in request data
- Range: Used to specify a between condition filter
- Operator: Indicates that the filter builder must copy over the MatchBy criteria specified in the incoming data
- AddOn: Indicates that the tag must simply be copied over as a filter criteria even if it not present in the incoming XML data

*Example 1:*

This is the raw XML as returned from a PGL:

```
<TO_DOCVAR AssignToVar="outXml">
<PARAMETERS>
<RECORD_COUNT Value="NaN"/>
<SYS_ID Value="C:/temp/SearchSchedule.pgl"/>
<MAX_ROWS Value="10"/>
<FROM_DELIVERY_DATE Value="03/24/2004"/>
<SORT_ORDER Value="Ascending"/>
<ORDER_POINT_ID Value=""/>
<PAGE Value="studio_ui"/>
<PAGE_NAME Value="studio_ui"/>
<STATE_ID Value=""/>
<DO_SEARCH Value="false"/>
<VISIBILITY Value="Simple"/>
<pagenum Value=""/>
<NO_OF_ROWS Value="0"/>
<TO_DELIVERY_DATE Value="03/24/2005"/>
<BUTTON_ID Value="search"/>
<SHIP_TO_NAME Value="*"/>
```

```
<SORT_BY Value="ID"/>
<START_COUNT Value="0"/>
<ITEM_ID Value=""/>
<SHIP_FROM_NAME Value="G*,P*"/>
<SHIP_FROM_NAME Value="*H*"/>
<SUBMIT_AS_XML Value="false"/>
<NUMBER Value="7800"/>
<NUMBER_MATCH_BY Value="GREATER_EQUAL"/>
</PARAMETERS>
</TO_DOCVAR>
```

This is the request to formulate the search filters from the XML data above:

```
<FORM_SEARCH_FILTER Value="{$outXml}" AssignToVar="filters">
<SHIP_TO_NAME/>
<SHIP_FROM_NAME MultiValued="true"/>
<STATE_ID MultiValued="true"/>
<ITEM_ID/>
<DELIVERY_DATE Range="true"/>
<NUMBER Operator="true"/>
<STATUS Value="ACTIVE" AddOn="true"/>
</FORM_SEARCH_FILTER>
```

This is the response:

```
<RESPONSE Status="Success">
<FILTERS>
<FILTER Name="SHIP_TO_NAME">
<SHIP_TO_NAME Value="%" MatchBy="LIKE"/>
</FILTER>
<FILTER Name="SHIP_FROM_NAME">
<OR>
<SHIP_FROM_NAME Value="G%" MatchBy="LIKE"/>
<SHIP_FROM_NAME Value="P%" MatchBy="LIKE"/>
<SHIP_FROM_NAME Value="%H%" MatchBy="LIKE"/>
</OR>
```

```
</FILTER>
<FILTER Name="DELIVERY_DATE">
<DELIVERY_DATE Value="03/24/2004" MatchBy="GREATER_EQUAL"/>
<DELIVERY_DATE Value="03/24/2005" MatchBy="LESS_EQUAL"/>
</FILTER>
<FILTER Name="NUMBER">
<NUMBER Value="7800" MatchBy="GREATER_EQUAL"/>
</FILTER>
<FILTER Name="STATUS">
<STATUS Value="ACTIVE" AddOn="true"/>
</FILTER>
</FILTERS>
</RESPONSE>
```

## FORM_GET

*Description:*

The `FORM_GET` tag helps in getting a list of nodes from another list based on a set of conditions. It can be used to select a set of nodes from a huge list using a filter.

*Syntax:*

```
<FORM_GET AssignToVar="var">
<FORM Value="{X-path expression}" ChildName="name"
KeyProp="property"/>
<FILTER SelectList="{X-path expression}"/>
</FORM_GET>
```

*Attributes:*

- AssignToVar (Required): Variable to assign the output to.
- FORM Value (Required): Parent node of the list from where the nodes have to be selected.
- FORM ChildName (Required): Child element name under the parent specified above.
- FORM KeyProp (Required): Key with which the filtering values would be matched.
- FILTER SelectList (Required): Item to be selected.

*Example 1:*

This is the list which needs to be filtered:

```
<WORLD>
<COUNTRIES>
```

```
<COUNTRY>
<NAME Value="India"/>
<CAPITAL Value="New Delhi"/>
<LANGUAGE Value="Hindi"/>
</COUNTRY>
<COUNTRY>
<NAME Value="USA"/>
<CAPITAL Value="Washington"/>
<LANGUAGE Value="English"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Russia"/>
<CAPITAL Value="Moscow"/>
<LANGUAGE Value="Russian"/>
</COUNTRY>
</COUNTRIES>
</WORLD>
```

This is the filter condition:

```
<PARAMETERS>
  <SELECTED Value="New Delhi"/>
  <SELECTED Value="Moscow"/>
</PARAMETERS>
```

This is the input:

```
<FORM_GET AssignToVar="outVar">
<FORM Value="{$myXml/COUNTRIES}" ChildName="COUNTRY"
KeyProp="CAPITAL"/>
<FILTER SelectList="$filter/SELECTED/@Value"/>
</FORM_GET>
<PRINTLN Value="{$outVar}"/>
```

This is the output:

```
<COUNTRY>
<NAME Value="India"/>
<CAPITAL Value="New Delhi"/>
```

```
<LANGUAGE Value="Hindi"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Russia"/>
<CAPITAL Value="Moscow"/>
<LANGUAGE Value="Russian"/>
</COUNTRY>
```

## FORM_REMOVE

*Description:*

The `FORM_REMOVE` tag helps in removing specific nodes from a list depending on a criterion. This tag can be used whenever you want to remove some node(s) from pipeline list depending on the set of user selections in a PGL UI table.

*Syntax:*

```
<FORM_REMOVE>
<FORM Value="{X-Path expression}" ChildName="child"
KeyProp="property"/>
<FILTER SelectList="filter"/>
</FORM_REMOVE>
```

*Attributes:*

- FORM Value (Required): Parent element in the pipeline variable where the removal operation has to be done.
- FORM ChildName (Required): Children to search for under the parent element.
- FORM KeyProp (Required): Key with respect to the xml list for child elements.
- `FILTER SelectList` (Required): List of nodes to be removed.

*Example 1:*

This is the node list:

```
<COUNTRY Value="India">
<POPULATION Value="80C"/>
<TOTAL_STATES Value="27"/>
<STATE Value="Maharashtra">
<CAPITAL Value="Bombay"/>
</STATE>
<STATE Value="Karnataka">
<CAPITAL Value="Bangalore"/>
```

```
</STATE>
<STATE Value="TamilNadu">
<CAPITAL Value="Chennai"/>
</STATE>
</COUNTRY>
```

This is the filter:

```
<PARAMETERS>
<CAP Value="Bombay"/>
<CAP Value="Chennai"/>
</PARAMETERS>
```

This is the input:

```
<FORM_REMOVE>
<FORM Value="{$myXml}" ChildName="STATE" KeyProp="CAPITAL"/>
<FILTER SelectList="$filter/CAP/@Value"/>
</FORM_REMOVE>
<PRINTLN Value="{$myXml}"/>
```

This is the output:

```
<COUNTRY Value="India">
<POPULATION Value="80C"/>
<TOTAL_STATES Value="27"/>
<STATE Value="Karnataka">
<CAPITAL Value="Bangalore"/>
</STATE>
</COUNTRY>
```

## FORM_UPDATE

*Description:*

The `FORM_UPDATE` action tag helps updating an xml variable from a flat xml structure. This tag can be used whenever you want to update a list based on the output from a PGL UI table. It does the collating of data from the and updating in one step.

*Syntax:*

```
<FORM_UPDATE>
<FILTER SelectList="list"/>
```

```
<FORM Value="{X-Path expression}" ChildName="child"
KeyProp="property"/>

<PROPS Source="{X-Path expression}">

<Property1/>

<Property2/>

........

</PROPS>

</FORM_UPDATE>
```

*Attributes:*

- FILTER SelectList (Required): List of values to be filtered from; it acts as keys to be searched in the xml to be updated.
- FORM KeyProp (Required): Key to use for filtering with respect to the tree xml list.
- FORM Value (Required): Parent element in the pipeline variable which needs to be updated
- FORM ChildName (Required): Children to search for under the parent specified by FORM Value attribute.
- PROPS Source (Required): The source xml which contains the new property values (say data from a UI table).

`FORM_UPDATE` first matches all the rows with the respective node in the list using FORM SelectList and FORM KeyProp attributes. After matching it updates the tags specified within `PROPS tag` from PROPS Source to the corresponding nodes in FORM ChildName.

*Example 1:*

The raw XML as returned from PGL to the workflow (contains correct values for the capital city and primary language):

```
<PARAMETERS>

<NAME Value="India"/>

<NAME Value="USA"/>

<CAPITAL Value="New Delhi"/>

<CAPITAL Value="Washinton DC"/>

<LANGUAGE Value="Hindi"/>

<LANGUAGE Value="English"/>

</PARAMETERS>
```

The xml to be updated (contains incorrect values):

```
<WORLD>

<COUNTRIES>
```

```
<COUNTRY>
<NAME Value="India"/>
<CAPITAL Value="Delhi"/>
<LANGUAGE Value="German"/>
</COUNTRY>
<COUNTRY>
<NAME Value="USA"/>
<CAPITAL Value="New York"/>
<LANGUAGE Value="Spanish"/>
</COUNTRY>
<COUNTRY>
<NAME Value="Russia"/>
<CAPITAL Value="Moscow"/>
<LANGUAGE Value="Russian"/>
</COUNTRY>
</COUNTRIES>
</WORLD>
```

The form update operation:

```
<FORM_UPDATE>
<FILTER SelectList="$formXml/NAME/@Value"/>
<FORM Value="{$oldXml/COUNTRIES}" ChildName="COUNTRY"
KeyProp="NAME"/>
<PROPS Source="{$formXml}">
<LANGUAGE/>
<NAME/>
<CAPITAL/>
</PROPS>
</FORM_UPDATE>
```

The updated xml is:

```
<WORLD>
  <COUNTRIES>
    <COUNTRY>
      <NAME Value="India"/>
```

```
      <CAPITAL Value="New Delhi"/>
      <LANGUAGE Value="Hindi"/>
    </COUNTRY>
    <COUNTRY>
      <NAME Value="USA"/>
      <CAPITAL Value="Washinton DC"/>
      <LANGUAGE Value="English"/>
    </COUNTRY>
    <COUNTRY>
      <NAME Value="Russia"/>
      <CAPITAL Value="Moscow"/>
      <LANGUAGE Value="Russian"/>
    </COUNTRY>
  </COUNTRIES>
</WORLD>
```

## TAG_ERRORS

*Description:*

`TAG_ERRORS` operation is primarily used in the UI workflows for grouping a set of errors under one element (or tag). It is very useful when you want to show an error icon beside a field whenever a set of error occurs. PGL shows error icon beside a field whenever it finds a _ERRORS tag under that particular field value.

*Syntax:*

```
<TAG_ERRORS SelectList="list" Name="property-name">
<Property1/>
<Property2/>
..........
</TAG_ERRORS>
```

*Attributes:*

- SelectList (Required): List of lines that needs to be searched for tagging set of errors.
- Name (Required): Tag under all the found errors are tagged

The tags under which error tag needs to search are specified within the component body.

*Example 1:*

This is the XML from which a set of errors is to be grouped under one tag:

```
<BPO>
  <SELLING_ORG_ID Value="Saturn"/>
  <SELLING_ORG_NAME Value="Saturn"/>
  <ORDER_POINT_ID Value="Velocity-Dallas"/>
  <ORDER_POINT_NAME Value="Velocity-Dallas"/>
  <BUYER_ORG_ID Value="Velocity"/>
  <TYPE_ID Value="QUANTITY"/>
  <NOTES/>
  <BPO_LINES>
    <BPO_LINE>
      <TYPE_ID Value="QUANTITY"/>
      <ID Value="bpoli-3311102557060272"/>
      <ITEM_ID Value="C-136D8710G01"/>
      <ITEM_DESC Value="Borescope Set - S"/>
      <SUPPLIER_PART_NUMBER/>
      <UOM Value="EA"/>
      <QTY_LIMIT Value="0">
        <_ERRORS Value="ERROR">
          <_ERROR Value="INVALID_VALUE" Severity="ERROR"/>
        </_ERRORS>
      </QTY_LIMIT>
      <QTY_USED Value="1">
        <_ERRORS Value="ERROR">
<_ERROR Value="QTY_LIMIT_LESS_THAN_QTY_USED" Severity="ERROR"/>
        </_ERRORS>
      </QTY_USED>
      <MIN_RELEASE_QTY Value="0"/>
      <MAX_RELEASE_QTY Value="0"/>
      <NOTIFY_QUANTITY_PERCENTAGE Value=""/>
    </BPO_LINE>
  </BPO_LINES>
</BPO>
```

This is the input operation:

```
<TAG_ERRORS SelectList="$currentBPO/BPO_LINES/BPO_LINE"
Name="QUANTITY_ERRORS">
  <QTY_LIMIT/>
  <QTY_USED/>
  <MIN_RELEASE_QTY/>
  <MAX_RELEASE_QTY/>
</TAG_ERRORS>
```

This is the output (you can see a new tag named `QUATITY_ERRORS` under `BPO_LINE` which has all the errors found under `QTY_LIMIT`, `QTY_USED`, `MIN_RELEASE_QTY` and `MAX_RELEASE_QTY`):

```
<BPO>
  <SELLING_ORG_ID Value="Saturn"/>
  <SELLING_ORG_NAME Value="Saturn"/>
  <ORDER_POINT_ID Value="Velocity-Dallas"/>
  <ORDER_POINT_NAME Value="Velocity-Dallas"/>
  <BUYER_ORG_ID Value="Velocity"/>
  <TYPE_ID Value="QUANTITY"/>
  <NOTES/>
  <BPO_LINES>
    <BPO_LINE>
      <TYPE_ID Value="QUANTITY"/>
      <ID Value="bpoli-3311102557060272"/>
      <ITEM_ID Value="C-136D8710G01"/>
      <ITEM_DESC Value="Borescope Set - S"/>
      <SUPPLIER_PART_NUMBER/>
      <UOM Value="EA"/>
      <QTY_LIMIT Value="0">
        <_ERRORS Value="ERROR">
          <_ERROR Value="INVALID_VALUE" Severity="ERROR"/>
        </_ERRORS>
      </QTY_LIMIT>
      <QTY_USED Value="1">
```

```
        <_ERRORS Value="ERROR">
          <_ERROR Value="QTY_LIMIT_LESS_THAN_QTY_USED"
Severity="ERROR"/>
        </_ERRORS>
      </QTY_USED>
      <MIN_RELEASE_QTY Value="0"/>
      <MAX_RELEASE_QTY Value="0"/>
      <NOTIFY_QUANTITY_PERCENTAGE Value=""/>
      <QUANTITY_ERRORS>
        <_ERRORS>
          <_ERROR Value="INVALID_VALUE" Severity="ERROR"/>
          <_ERROR Value="QTY_LIMIT_LESS_THAN_QTY_USED"
Severity="ERROR"/>
        </_ERRORS>
      </QUANTITY_ERRORS>
    </BPO_LINE>
  </BPO_LINES>
</BPO>
```

## DROP_DOWN_OPTIONS

**Note:** Not functional in MDM 1.00.01

*Description:*

This action tag is primarily used in the UI workflows for generating options in the format that the PGL ui:dropdown tag requires. It is mostly used in preprocessing stage of a UI node in a workflow.

*Syntax:*

```
<DROP_DOWN_OPTIONS SelectList="$orderPointInfo/SHIP_TO_LIST/SHIP_TO"
Id="ID" Value="NAME" FirstId="{$orderPointInfo/SHIP_TO/ID/@Value}"/>
```

*Attributes:*

- SelectList (Required): List of values that needs to displayed in the UI dropdown box.
- Id (Required): Tag that corresponds to your dropdown box IDs.
- Value (Required): Tag that corresponds to your dropdown box values.
- FirstId (Optional): The value which becomes the first option in the output options.

*Example 1:*

This is the XML from which the drop down options has to be generated:

```
<ORDER_POINT_INFO Status="Success">
  <SHIP_TO_LIST>
    <SHIP_TO>
      <ID Value="Velocity-Warren"/>
      <NAME Value="Velocity-Warren"/>
    </SHIP_TO>
    <SHIP_TO ExistingDocument="yes">
      <ID Value="Velocity-DC1"/>
      <NAME Value="Velocity-DC1"/>
    </SHIP_TO>
    <SHIP_TO ExistingDocument="yes">
      <ID Value="Velocity-DC2"/>
      <NAME Value="Velocity-DC2"/>
    </SHIP_TO>
  </SHIP_TO_LIST>
</ORDER_POINT_INFO>
```

This is the input command:

```
<DROP_DOWN_OPTIONS SelectList="$myXml/SHIP_TO_LIST/SHIP_TO" Id="ID"
Value="NAME" FirstId="{$myXml/SHIP_TO/ID/@Value}"/>
```

The options list output below can be used as the input to the PGL UI dropdown tag optionSrc attribute:

```
<OPTION Value="Velocity-Warren" Id="Velocity-Warren"/>
<OPTION Value="Velocity-DC1" Id="Velocity-DC1"/>
<OPTION Value="Velocity-DC2" Id="Velocity-DC2"/>
```

## CREATE URL

*Description:*

This action tag is used for creating URLs based on the child elements to the tag.

*Syntax:*

```
<CREATE_URL Name="literal or {xpath}" AssignToVar="myUrl">
<XYZ Value="aaaa"/>
<ABC Value="bbbb"/>
<TO_XML SelectList="$a/*"/>
```

```
</CREATE_URL>
```

*Attributes:*

- Name (Required): The base url to which the values need to be appended.
- AssignToVar (Required): Variable to which the resultant url string is assigned.

*Example 1:*

Following is example of how the tag can be used inside an x-rule:

```
<CREATE_URL Name="{$webuiContextPath}/cpm/planning/tabs/
showCurrentForecastByEventType.cmd?" AssignToVar="myUrl">
<SUPPLIER_ID Value="{$thisParam/SUPPLIER_ID/@Value}"/>
<ITEM_ID Value="{$thisParam/ITEM_ID/@Value}"/>
<CUSTOMER_ID Value="{$thisParam/CUSTOMER_ID/@Value}"/>
</CREATE_URL>
```

When passed a thisParam input as follows:

```
<….>
<SUPPLIER_ID Value="sup1"/>
<ITEM_ID Value="item1"/>
<CUSTOMER_ID Value="cust1"/>
</…>
```

The following output is produced:

```
//localhost:7001/omxclient/cpm/planning/
showCurrentForecastByEventType.cmd?SUPPLIER_ID=sup1&ITEM_ID=item1&CU
STOMER_ID=cust1
```

# Utility Operations

## START_WORKFLOW

*Description:*

This operation can be used to initiate a MDM workflow.

*Syntax:*

```
<START_WORKFLOW>
<TEMPLATE Value="workflowName"/>
<ADDITIONAL_PARAMETERS>
        <childTag Value="value"/>
        </ADDITIONAL_PARAMETERS>
```

```
</START_WORKFLOW>
```

*Child Tags/Attributes:*

- TEMPLATE, Value (Required): Name of the workflow to be run.
- ADDITIONAL_PARAMETERS (Optional): Use this tag to pass additional parameters to the workflow. These parameters will be available in the service variable 'thisParam' in the Start node of the new workflow.

*Example:*

```
<START_WORKFLOW>
<TEMPLATE Value="newPartRequest"/>
<ADDITIONAL_PARAMETERS>
        <ItemID Value="RE-10391"/>
        </ADDITIONAL_PARAMETERS>
</START_WORKFLOW>
```

## PRINT/PRINTLN

*Description:*

The `PRINT` , `PRINTLN` statements are very useful for debugging X-Rules. By default, PRINT publishes the value of the LhsExpression on to the console (or a log file, if it is specified) when the log level is set to Debug. However, when the optional attribute ToVar is used with PRINT, the value of the LhsExpression is concatenated on to the varName variable specified by ToVar.

`PRINTLN` has the same effect as `PRINT` except that it also prints a new line.

*Syntax:*

```
<PRINT LhsExpression {ToVar="varName"}/>
```

OR

```
<PRINTLN LhsExpression {ToVar="varName"} />
```

*Attributes:*

- Select (Optional): Variable to be printed.
- SelectList (Optional): Variable list to be printed.
- ToVar (Optional): If specified, the value of the LhsExpression is concatenated to this variable.
- Value (Optional): X-Path expression to select values to be printed.

Exactly one of these attributes - Select, SelectList, Value - should be specified.

*Example 1:*

Input:

```
<SET Var="myQuantity" FromValue="50"/>
```

```
<PRINT Value="The Value of My Quantity is:{$myQuantity} "/>
```

Output:

```
The Value of My Quantity is: 50
```

*Example 2:*

Input:

```
<SET Var="myQuantity" FromValue="50"/>

<PRINTLN Value=" Printing to a string: " ToVar="MyPrintVar"/>

<PRINT Value="The Value of My Quantity is:{$myQuantity} "
ToVar="MyPrintVar"/>

<PRINTLN Value="{$MyPrintVar}"/>
```

Output:

Printing to a string:

```
The Value of My Quantity is: 50
```

## LOG

**Note:** Not supported in 1.00.01

*Description:*

The LOG operation is used to write to the log file. It is similar to the `PRINT/PRINTLN` operations but with the advantage that you can control the log level.

*Syntax:*

```
<LOG Loglevel= "level" Value="Log Statement"/>
```

*Attributes:*

- Loglevel (Required): One of - Error, Exception, Debug, Info, Warning, Audit, EnterTrace, ExitTrace, IntraServiceCallTrace, InterServiceCallTrace, SqlCallTrace, XPathFnTrace
- Value (Required): Text to be written to the log file

*Example 1:*

```
<LOG Loglevel="Info" Value="Write this in log file. Document is
{$doc}"/>
```

## EXECUTE_SHELL_COMMAND

*Description:*

You can perform OS Shell Commands and get the results in a XML format. The calling application is responsible for issuing platform dependent command statement. There is no restriction on what command can be issued, and hence you should take care to not use a command that can hang the X-Server!

*Syntax:*

```
<EXECUTE_SHELL_COMMAND Value= "{x-path expression}"
AssignToVar="var"/>
```

*Example 1:*

The following command will return the default Java version for the system:

```
<EXECUTE_SHELL_COMMAND Value="java -version" AssignToVar="response"/
>
```

The following response is returned for the above operation:

```
<RESPONSE Status="Success" SHELL_RETURN_VALUE="0">

<STD_OUT>

<LINE Value="java version &amp;quot;1.3.1_01&amp;quot;"/>

<LINE Value="Java(TM) 2 Runtime Environment,

Standard Edition (build 1.3.1_01)"/>

<LINE Value="Java HotSpot(TM) Client VM (build 1.3.1_01, mixed
mode)"/>

</STD_OUT>

      <STD_ERR/>

</RESPONSE>
```

*Example 2:*

Adding TimeOut Attribute for the EXECUTE_SHELL_COMMAND

Adding TimeOut Attribute for the EXECUTE_SHELL_COMMAND in general provides a timeout for execution of any script. You can specify TimeOut in seconds for providing an execution timeout for your shell command execution as the value for this parameter. If the execution time exceeds timeout ShellReturnValue attribute if set to 1 and ExecutionTimedout attribute is set to "yes" in the NodeOutput response.

Sample timed out response for script execution:

```
<RESPONSES Status="Success">

  <RESPONSE ExecutionTimedout="yes" Status="Success"
ShellReturnValue="1">

    <STD_OUT>

      <LINE Value="sleeping before execution..."/>

    </STD_OUT>

    <STD_ERR/>

  </RESPONSE>

</RESPONSES>
```

## STORE_XML_FILE

*Description:*

This action tag will write the XML representation of an XMLform instance to the specified file on the local file system.

*Syntax:*

```
<STORE_XML_FILE FromSelect="$xml" Var="fileName" Beautify="true/
false"/>
```

*Attributes:*

- FromSelect (Required): Variable representing XML
- Var (Required): File name
- Beautify (Optional): true/false. When the attribute exists and the value is true the xml written out will be formatted. Default value of the attribute (when it is not explicitly specified) is false.

*Example 1:*

```
<STORE_XML_FILE FromSelect="$xml" Var="myFile1.xml" Beautify="true"/
>
```

## PRINT_TO_FILE

*Description:*

This action tag will write the string representation of any object onto a file on the local file system. It can also be used to append text to an existing file.

*Syntax:*

```
<PRINT_TO_FILE File="filename" NewFile="yes/no" SameLine="yes/no"
Value="text" Beautify="true/false"/>
```

*Attributes:*

- File (Required): A file in a valid directory of the file system. Directories would NOT be created. "No directory", would imply current directory. If such a file by that name does not exist, then a file would be created.
- NewFile (Optional): 'yes/no'. A "yes" would imply that a new file needs to be created with the name. If there is a file already existing with the name, then the file would be cleaned up and writing would start at the Top. A "no" would imply that writing would continue at the bottom of the File; a new File will be created if there is no such File present in the given directory. Default value is "no".
- SameLine (Optional): 'yes/no'. A "yes" would imply that writing need to continue on the same Line. A "no" would imply that writing would continue on a new Line. Default value is "no".
- Value (Required): An X-Path expression to represent the content for the file.

- Beautify (Optional): true/false. When the attribute exists and the value is true the output will be formatted if the entire content of string written out is parseable xml content. Default value of the attribute (when it is not explicitly specified) is false.

*Example 1:*

```
<PRINT_TO_FILE File="D:\\Temp\\myFile.txt" NewFile="yes"
SameLine="yes" Value="{$test} {$work} all the time {$product}"
Beautify="true"/>
```

## GET_CURRENT_TIME

*Description:*

This action tag return the current time. If a logical date is set, then current time is as per the logical date.

*Syntax:*

```
<GET_CURRENT_TIME AssignToVar="myTime"/>
```

*Attributes:*

AssignToVar (Required): Variable to assign the current time to

*Example 1:*

```
<GET_CURRENT_TIME AssignToVar="CurrentTime"/>
```

## SET_LOGICAL_DATE

*Description:*

This action tag is used to set the logical date of the application server.

*Syntax:*

```
<SET_LOGICAL_DATE Value="X-Path function"/>
```

*Attributes:*

Value (Required): X-Path date function or date string

*Example 1:*

```
<SET_LOGICAL_DATE Value="{incrDate(date(), duration(0,(-1),0,0))}"/>
```

## HANDLE_SYSTEM_EXCEPTION

*Description:*

This statement is used within `DEFINE_METHOD` tags to catch exceptions related to `Studio/Database/Remote/Java`. This statement will not catch user exceptions thrown within the body of `DEFINE_METHOD`. Any action statement can be provided within this tag. There is full access to thisParam, thisReturn and other variables created before the exception was encountered.

This tag is provided only for doing cleanup and throwing a well-formatted exception. If no exception is thrown within `HANDLE_SYSTEM_EXCEPTION`, then the original exception will be thrown back. There is no real way to suppress this exception. Also, there can be only one `HANDLE_SYSTEM_EXCEPTION` within a method definition that will handle all exceptions. This should be a direct child of `DEFINE_METHOD`.

*Syntax:*

```
<DEFINE_METHOD>
.........
<HANDLE_SYSTEM_EXCEPTION>
<DEFINE_METHOD>
```

*Attributes:*

None

*Example:*

```
<DEFINE_METHOD Name="exceptionTest" >
<RULE>
<ACTION>
<REQUEST Name="remoteProcess" AssignToVar="result"/>
</ACTION>
</RULE>
<HANDLE_SYSTEM_EXCEPTION>
<ACTION>
<EXCEPTION>
<_RESULT Value="SEVERE_ERROR"/>
<_ERROR Value="Exception in Remote Process"/>
</EXCEPTION>
</ACTION>
</HANDLE_SYSTEM_EXCEPTION>
</DEFINE_METHOD>
```

## EXCEPTION

*Description:*

Exceptions are way to communicate with an external module, that an abnormal condition has been encountered. This results in a rollback of the transaction and returns a well-structured XML response. An Exception can be raised inside an X-Rule; however, it is not possible to handle and recover from Exceptions inside the X-Rule.

When the rule interpreter encounters an Exception statement, it stops interpreting the rules and propagates the exception to the caller. The entire XML contained inside `EXCEPTION` tag will be reported as a part of the error message.

*Syntax:*

```
<EXCEPTION {Value="xxx"}>
<Any XML Structure>
</EXCEPTION>
```

*Attributes:*

Value (Required): Error message

*Example:*

The following statement exits the rule module by throwing an exception. This exception is an instance com.teradata.X-Core.util.X-CoreException.

```
<DEFINE_METHOD Name="exceptionTest">
<RULE>
<ACTION>
<IF_TEST Test="$thisParam/ID/@Value=null">
<THEN>
<EXCEPTION Value="MISSING_MANDATORY_FIELD">
<ARG0 Value="ID" />
</EXCEPTION>
</THEN>
<ELSE>
<SET DocVar="thisReturn" Property="IS_VALID" FromValue="TRUE"/>
</ELSE>
</IF_TEST>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

If the input request is:

```
<REQUESTS ServiceName="XXXX">
<REQUEST Name="exceptionTest"/>
</REQUESTS>
```

Then the response will be:

```
<RESPONSES Status="Error">
<RESPONSE Description="MISSING_MANDATORY_FIELD" Status="Error">
<ARG0 Value="ID"/>
</RESPONSE>
</RESPONSES>
```

## DO_TRANSACTION

*Description:*

This action is used when the user wants to execute a new transaction in the same thread. This means that all action tags defined as children of `DO_TRANSACTION` will be executed in a different context by the same java thread invoking the `DO_TRANSACTION` action.

*Syntax:*

```
<DO_TRANSACTION>
{One or more Operations}
</DO_TRANSACTION>
```

*Attributes:*

None

*Example 1:*

```
<DO_TRANSACTION>
<REQUEST Name="updateOrder">
<CUSTOMER_ORDER_ID Value="87"/>
<MODIFIED_DATE Value="2/23/2004"/>
</REQUEST>
</DO_TRANSACTION>
```

## REFRESH_FROM_CACHE

*Description:*

If you know that a document instance is in cache and you have the instance reference in X-Rules, you can boost system performance by refreshing the value of properties of your instance by invoking the `REFRESH_FROM_CACHE` operation that accesses the MDM cache directly and updates the values of the instance in the X-Rules.

The document instance is accessed using the root tag name which maps to document name and the primary keys that are available in the instance specified. If the instance is not available in MDM cache, then an exception is thrown.

*Syntax:*

```
<REFRESH_FROM_CACHE Select="$myVar">
<PROP Name="Property1"/>
<PROP Name="Property2"/>
.............
</REFRESH_FROM_CACHE>
```

*Attributes:*

Select (Required): XML document variable

*Example 1:*

In the example below, the specified property values will be obtained from cache and updated on the XML referred to by variable mySched:

```
<REFRESH_FROM_CACHE Select="$mySched">
<PROP Name="SHIP_FROM_ID"/>
<PROP Name="SHIP_DATE"/>
<PROP Name="QTY"/>
<PROP Name="NON_EXISTING_PROP"/>
<PROP Name="CITY_TAX"/>
</REFRESH_FROM_CACHE>
```

## EXEC_INFORMATICA

*Description:*

The `EXEC_INFORMATICA` tag can be used to invoke an Informatica transform directly from MDM. The assumption is that Informatica client is installed on the same machine where the MDM server is installed.

**Note:** To use this tag, ensure that your X-Service file has the following configuration:

- The extension rule file 'bcm_rule_extension.xml' is specified. For e.g.:

  ```
  <extensionFiles>
  <extensionFile Name="com/teradata/bcm/xcore/serverutil/xml/
  bcm_rule_extension.xml"/>
  </extensionFiles>
  ```

- The `<extensionFiles>` block occurs before the `<ruleDefnFiles>` block. For e.g.:

  ```
  <dom-config>
  ......
  <extensionFiles>
  ......
  </extensionFiles>
  <ruleDefnFiles>
  ```

```
......
</ruleDefnFiles>
......
</dom-config>
```

- The service parameters specify other parameters needed to connect to Informatica:

```
<service-params xmlns="">
<param xmlns="" Value="&amp;quot;C:\Program
Files\Informatica\PowerCenter Server\bin\PmCmd&amp;quot;"
Name="pmCmdPath"/>
<param xmlns="" Value="Administrator" Name="repUserName"/>
<param xmlns="" Value="Administrator" Name="repPassword"/>
<param xmlns="" Value="blrqa01" Name="hostName"/>
<param xmlns="" Value="4001" Name="portNo"/>
</service-params>
```

  Where '`pmCmdPath`' parameter points to the bin folder of Informatica's PowerCenter.

*Syntax:*

```
<EXEC_INFORMATICA AssignToVar="response">
<FOLDER Value="informatica-folder-name"/>
<WORKFLOW Value= "informatica-workflow-name"/>
<BATCH Value="informatica-batch-name "/>
<SESSION Value="informatica-session-name"/>
<WAIT_FLAG Value="0/1"/>
</EXEC_INFORMATICA>
```

*Attributes:*

AssignToVar (Optional): Variable to which the response is assigned

*Child Tags:*

- FOLDER (Required): Folder where the workflow/batch/session file is located
- WORKFLOW (Optional): Informatica Workflow name
- BATCH (Optional): Informatica Batch name
- SESSION (Optional): Informatica Session name
- WAIT_FLAG (Required): "0/1"

**Note:** Exactly one of the three - WORKFLOW, BATCH, and SESSION - should be provided.

*Example:*

```
<EXEC_INFORMATICA AssignToVar="response">
<FOLDER Value="custdb_input"/>
<BATCH Value="b_all"/>
```

```
<WAIT_FLAG Value="1"/>

</EXEC_INFORMATICA>
```

## EXECUTE_TRILLIUM

*Description:*

The EXECUTE_TRILLIUM tag can be used to invoke Trillium Data Matching and Cleansing directly from MDM. The assumption is that Trillium client is installed.

**Note:** To use this command tag, the "Trillium package" must be included in the external library files.

Use of this method requires a special configuration file to define the format of the XML document. Use the TRILLCONFIG environment variable to point to this configuration file.

The current tested version is Trillium V.11

*Syntax (For cleanse):*

```
<EXECUTE_TRILLIUM Type="Cleanse" SYS_ID="G" server Name="Cleanser"
RootTag="CustomerData/PrimaryCustomer/Address">
<!—User input XML starts here->
<CustomerData>
<REQUESTTYPE> Cleanse </REQUESTTYPE>
      <PrimaryCustomer>
         <Name>CARLY CHRIS</Name>
         <Address>
            <StreetLine1>109 PERKINS AVE</StreetLine1>
            <StreetLine2 />
            <City>LEESBURG</City>
            <State>FL</State>
            <Country>US</Country>
            <ZipCode>34748</ZipCode>
         </Address>
      </PrimaryCustomer>
</CustomerData>
<!—User input XML finishes here->
</EXECUTE_TRILLIUM>
```

*Attributes:*

- **Type**: The Type attribute of this command may values, "Cleanse", “Match” or "MatchSuspect", depending on which cleaning or appropriate Matching action will be taken.
- **SYS_ID**: Specifies the System ID of requested server - which means that it is a Registered Cleanser or Matcher System ID. It is required to connect a Director Instance.
- **ServerName**: Should be the name of the registered Trillium server which is attached to the System ID. It is a Registered Cleanser or Matcher Server name. It is required to connect to a Director Instance.
- **RootTag**: It is the XML path expression for the data you are providing as an input.
- **AssignToVar** (Optional): The variable "outputResult" specified in AssignToVar will contain the results returned after cleanse/match operation get performed.
- **Child Tags**:

- **<REQUESTTYPE/>**: This tag should contain the operation you want to perform. It should have values “cleanse”,”match”or “suspectmatching”.

## Example

1 Cleansing:

```
<EXECUTE_TRILLIUM Type="Cleanse" SYS_ID="G" server Name="Cleanser"
RootTag="CustomerData/PrimaryCustomer/Address">
<!—User input XML starts here->
<CustomerData><REQUESTTYPE>Cleanse</REQUESTTYPE>
<PrimaryCustomer>
    <Name>Alan Wolfson</Name>
    <Address>
       <StreetLine1>25 Linnell Circle</StreetLine1>
       <StreetLine2 />
       <City>Billerica</City>
       <State>MA</State>
       <Country>US</Country>
       <ZipCode></ZipCode>
       <WINDOW_KEY_01></WINDOW_KEY_01>
       <NEWADDRL1></NEWADDRL1>
       <PR_NAME_GENDER_01></PR_NAME_GENDER_01>
       <US_GOUT_STREET_SUFFIX></US_GOUT_STREET_SUFFIX>
       <US_CEN_RESLV_LONGITUDE></US_CEN_RESLV_LONGITUDE>
       <US_CEN_RESLV_LATITUDE></US_CEN_RESLV_LATITUDE>
    <US_CEN_RESLV_COORD_LEVEL></US_CEN_RESLV_COORD_LEVEL>
       <US_GOUT_HOUSE_NUMBER></US_GOUT_HOUSE_NUMBER>
       <US_GOUT_STREET_NAME></US_GOUT_STREET_NAME>
    <US_GOUT_POSTAL_CITY_NAME></US_GOUT_POSTAL_CITY_NAME>
       <US_GOUT_STATE_NAME></US_GOUT_STATE_NAME>
       <US_GOUT_POSTAL_CODE></US_GOUT_POSTAL_CODE>
       <US_GOUT_MATCH_LEVEL></US_GOUT_MATCH_LEVEL>
       <US_GOUT_RECORD_TYPE></US_GOUT_RECORD_TYPE>
       <NEWADDRL2></NEWADDRL2>
       <NEWADDRL6></NEWADDRL6>
       <NEWADDRL7></NEWADDRL7>
<PR_NAME_SUFFIX_RECODED_01></PR_NAME_SUFFIX_RECODED_01>
<PR_GIVEN_NAME1_RECODED_01></PR_GIVEN_NAME1_RECODED_01>
<PR_SURNAME1_RECODED_01></PR_SURNAME1_RECODED_01>
<PR_NAME_FORM_01></PR_NAME_FORM_01>
</Address>
</PrimaryCustomer>
</CustomerData>
<!—User input XML finishes here->
</EXECUTE_TRILLIUM>
```

*Expected Output:*

```
<CustomerData>
<REQUESTTYPE>Cleanse</REQUESTTYPE>
<PrimaryCustomer>
    <Name>Alan Wolfson</Name>
    <Address>
       <StreetLine1>25 LINNELL CIR</StreetLine1>
       <StreetLine2/>
       <City>BILLERICA</City>
       <State>MA</State>
       <Country>United States</Country>
       <ZipCode/>
```

```
        <WINDOW_KEY_01>018WLLN1</WINDOW_KEY_01>
        <NEWADDRL1>ALAN WOLFSON</NEWADDRL1>
        <PR_NAME_GENDER_01>M</PR_NAME_GENDER_01>
        <US_GOUT_STREET_SUFFIX>CIR</US_GOUT_STREET_SUFFIX>
<US_CEN_RESLV_LONGITUDE>071251143</US_CEN_RESLV_LONGITUDE>
<US_CEN_RESLV_LATITUDE>+42527873</US_CEN_RESLV_LATITUDE>
<US_CEN_RESLV_COORD_LEVEL>1</US_CEN_RESLV_COORD_LEVEL>
<US_GOUT_HOUSE_NUMBER>25</US_GOUT_HOUSE_NUMBER>
<US_GOUT_STREET_NAME>LINNELL</US_GOUT_STREET_NAME>
<US_GOUT_POSTAL_CITY_NAME>BILLERICA</US_GOUT_POSTAL_CITY_NAME>
<US_GOUT_STATE_NAME>MA</US_GOUT_STATE_NAME>
<US_GOUT_POSTAL_CODE>018213928</US_GOUT_POSTAL_CODE>
<US_GOUT_MATCH_LEVEL>0</US_GOUT_MATCH_LEVEL>
<US_GOUT_RECORD_TYPE>S</US_GOUT_RECORD_TYPE>
<NEWADDRL2>25 LINNELL CIR</NEWADDRL2>
<NEWADDRL6>MA, 01821-3928</NEWADDRL6>
<NEWADDRL7>WOLFSON ALAN</NEWADDRL7>
<PR_NAME_SUFFIX_RECODED_01></PR_NAME_SUFFIX_RECODED_01>
<PR_GIVEN_NAME1_RECODED_01>ALAN</PR_GIVEN_NAME1_RECODED_01    >
<PR_SURNAME1_RECODED_01>WOLFSON</PR_SURNAME1_RECODED_01>
<PR_NAME_FORM_01/>
</Address>
</PrimaryCustomer>
</CustomerData>
```

**2** Matching:

```
<EXECUTE_TRILLIUM Type="MatchSuspect" SYS_ID="G"
serverName="RMatcher" RootTag="CustomerData/PrimaryCustomer">
<! -- This is the input (dirty) XML document -->

<CustomerData>
<REQUESTTYPE>RMatch</REQUESTTYPE>
<PrimaryCustomer>
    <Name>MR.SMITH</Name>
    <CUST_ID>281679</CUST_ID>
    <SRC_SYSTEM_ID>ERP-DB</SRC_SYSTEM_ID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
       <Country>US</Country>
       <ZipCode>347368502</ZipCode>
    </Address>
</PrimaryCustomer>
<Candidate>
    <Name> MR.SMITH </Name>
    <PartyID>894</PartyID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
       <Country>US</Country>
       <ZipCode>347368502</ZipCode>
    </Address>
</Candidate>
</CustomerData>
</EXECUTE_TRILLIUM>
```

*Expected Output Will Be:*

```
<CustomerData>
<REQUESTTYPE>RMatch</REQUESTTYPE>
<PrimaryCustomer>
    <Name> MR.SMITH </Name>
    <CUST_ID>281679</CUST_ID>
    <SRC_SYSTEM_ID>ERP-DB</SRC_SYSTEM_ID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
       <Country>US</Country>
       <ZipCode>347368502</ZipCode>
    </Address>
</PrimaryCustomer>
<Candidate matchPattern="102">
    <Name> MR.SMITH </Name>
    <PartyID>894</PartyID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
       <Country>US</Country>
       <ZipCode>347368502</ZipCode>
    </Address>
</Candidate>
</CustomerData>
```

**3** Matching WithSupspect:

```
<EXECUTE_TRILLIUM Type="MatchSuspect" SYS_ID="G"
serverName="RMatcher" RootTag="CustomerData/PrimaryCustomer">
<! -- This is the input (dirty) XML document -->

<CustomerData>
<REQUESTTYPE>RMatch</REQUESTTYPE>
<PrimaryCustomer>
    <Name>MR.SMITH</Name>
    <CUST_ID>281679</CUST_ID>
    <SRC_SYSTEM_ID>ERP-DB</SRC_SYSTEM_ID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
       <Country>US</Country>
       <ZipCode>347368502</ZipCode>
    </Address>
</PrimaryCustomer>
<Candidate>
    <Name> MR.SMITH </Name>
    <PartyID>894</PartyID>
    <Address>
       <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
       <StreetLine2/>
       <City>GROVELAND</City>
       <State>FL</State>
```

```
        <Country>US</Country>
        <ZipCode>347368502</ZipCode>
    </Address>
</Candidate>
</CustomerData>
</EXECUTE_TRILLIUM>
```

*Expected Output Will Be:*

```
<CustomerData>
<REQUESTTYPE>RMatch</REQUESTTYPE>
<PrimaryCustomer>
    <Name> MR.SMITH </Name>
    <CUST_ID>281679</CUST_ID>
    <SRC_SYSTEM_ID>ERP-DB</SRC_SYSTEM_ID>
    <Address>
        <StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
        <StreetLine2/>
        <City>GROVELAND</City>
        <State>FL</State>
        <Country>US</Country>
        <ZipCode>347368502</ZipCode>
    </Address>
</PrimaryCustomer>
<Candidate matchPattern="102">
<Name>KARUU MOHAN</Name>
<PartyID>894</PartyID>
<Address>
<StreetLine1>20545 INDEPENDENCE BLVD</StreetLine1>
<StreetLine2/>
<City>GROVELAND</City>
<State>FL</State>
<Country>US</Country>
<ZipCode>347368502</ZipCode>
</Address>
</Candidate>
</CustomerData>
```

**Note:** Trillium Java API with XML type input expects above mentioned input data format which is Trillium Standards. All the above outputs are based on the configuration file (Trilxml.cfg).

## NATIVE_FUNCTION

*Description:*

The `NATIVE_FUNCTION` statement is analogous to the X-PATH nativeFunction(), which is the recommended approach to use Java Functions (see Chapter 10: "X-Path" for details on defining and using this type of function). The statement's syntax is provided here only for reference.

*Syntax:*

```
<NATIVE_FUNCTION ClassName="myClassName" MethodName="myMethodName"
{AssignToVar="rtnVar"}>

<PARAM LhsExpression1/>

.............
```

```
.............
<PARAM LhsExpressionN />
</NATIVE_FUNCTION >
```

Where:

- `myClassName`: Class Name
- `myMethodName`: Method Name
- `rtnVar`: Variable to store the return value; Optional
- `LhsExpression1` - LhsExpressionN: Parameters

*Example:*

```
<NATIVE_FUNCTION
ClassName="com.teradata.xcore.idgen.UniqueIdGenerator"
MethodName="getUniqueId" AssignToVar="myId">
<PARAM Value="PO"/>
</NATIVE_FUNCTION>
```

## INSTANCE_FUNCTION

*Description:*

The `INSTANCE_FUNCTION` statement is analogous to the X-PATH `instanceFunction()`, which is the recommended approach to use Java Functions (see Chapter 10: “X-Path” for details on defining and using this type of function). The statement's syntax is provided here only for reference.

*Syntax:*

```
<INSTANCE_FUNCTION ClassName="myClassName" MethodName="myMethodName"
{AssignToVar="rtnVar"}>
<PARAM LhsExpression1/>
.............
.............
<PARAM LhsExpressionN />
</INSTANCE_FUNCTION >
```

Where:

- `myClassName`: Class Name
- `myMethodName`: Method Name
- `rtnVar`: Variable to store the return value; Optional
- `LhsExpression1` - LhsExpressionN: Parameters

*Example:*

```
<INSTANCE_FUNCTION
ClassName="com.teradata.xcore.idgen.UniqueIdGenerator"
MethodName="getUniqueId" AssignToVar="myId">

    <PARAM Value="PO"/>

</INSTANCE_FUNCTION>
```

## INSTANCE_METHOD

*Description:*

The `INSTANCE_METHOD` statement is analogous to the X-PATH instanceMethod(), which is the recommended approach to use Java Functions (see Chapter 10: "X-Path" for details on defining and using this type of function). The statement's syntax is provided here only for reference.

*Syntax:*

```
<INSTANCE_METHOD MethodName="myMethodName" ObjVar="myObjVar"
{AssignToVar="rtnVar"}>

<PARAM LhsExpression1/>

.............

.............

<PARAM LhsExpressionN />

</INSTANCE_METHOD>
```

Where:

- `myMethodName`: Method Name
- `myObjVar`: Object to invoke the method on
- `rtnVar`: Variable to store the return value; Optional
- `LhsExpression1` - LhsExpressionN: Parameters

*Example:*

```
<INSTANCE_METHOD MethodName="getTotalEstimatedInvoiceValue"
ObjVar="coObj" AssignToVar="invoiceValue"/>
```

## NOTIFY

*Description:*

`Notify` statement is used for STEP level notification to monitor lowest level of logical unit of work as required in the workflow.

*Syntax:*

```
<NOTIFY StepName="StepName" StepType="ENTRY|EXIT|EVENT"
Message="Custom Message"/>
```

*Attributes:*

StepName—is the required, unique and user defined name for STEP. StepName must be unique across the underlying workflow.

StepType—is the required attribute for STEP and value would be ENTRY, EXIT or EVENT.

Message—is the optional attribute to specify any user defined message for the STEP.

For every STEP of type ENTRY and/or EXIT should match with the corresponding ENTRY and/or EXIT type of STEP for the underlying STEP. Also, the step name should be same for the ENTRY/EXIT pair. EVENT type is one time notification which can be used to notify any specific event in the workflow.

*Example:*

```
<NOTIFY StepName="E2ENode-FetchDataStep" StepType="ENTRY"
Message="NumberOfRowsRequired=100, TableName=AttributeSet,
GlobalValue=941"/>
```

### EVAL_EXPR

*Description:*

`EVAL_EXPR` statement is a convenient way of using the nativeFunction, instanceFunction and instanceMethod X-Path functions for calling Java methods that have a return type of void or when calling a method with a non-void return type while dropping the return value.

**Note:** The SET statement should be used to assign the return value to a variable when calling a Java method with a non-void return type.

*Syntax:*

```
<EVAL_EXPR Value="{nativeFunction(...) | instanceMethod(...) |
instanceFunction(...)}"/>
```

*Attributes:*

Value (Required): Function call

*Example:*

```
<EVAL_EXPR Value="{instanceMethod('MsgHeader', 'validate',
$msgHeader)}"/>
```

# Framework Service Operations

### VALIDATE_DOC

*Description:*

This tag is used to validate a given XML against one or more validation specifications. The action statement will stop validating and return upon encountering a severe error. For other errors and successes, it will continue executing all the specified validation specs and finally return an aggregated result. This action tag will work based on XMLform object reference.

*Syntax:*

```
<VALIDATE_DOC Value="{var}">
<SPEC Name="validateSpec1"/>
<SPEC Name="validateSpec2"/>
.....
</VALIDATE_DOC>
```

*Example:*

```
<VALIDATE_DOC Value="{$ship}">
<SPEC Name="baseShipmentValidation"/>
<SPEC Name="sentShipment"/>
</VALIDATE_DOC>
```

If the `$ship` variable referred to the following document structure,

```
<SHIP>
......
</SHIP>
```

The response after execution of validation will be like:

```
<SHIP ValidationResult="SUCCESS">
.....
</SHIP>
```

The other values possible for ValidationResult attribute are `ERROR` and `SEVERE_ERROR`.

## UPDATE_VALIDATION_RESULT

*Description:*

This action statement is related to the previous `VALIDATE_DOC` statement. In the previous example if `SHIP` had children `SHIP_LINES` and the `SHIP_LINES` were validated through different specifications, one might want to aggregate the two results obtained. This action statement will compare the ValidationResult attributes of the 2 document structures and indicate a single validation result on the `SHIP` doc.

The order of preference will be SEVERE_ERROR > ERROR > SUCCESS

*Syntax:*

```
<UPDATE_VALIDATION_RESULT Value="{$ship}" FromValue="{$shipLine}"/>
```

*Example:*

If `$ship` refers to

```
<SHIP ValidationResult="SUCCESS">
```

```
…..
</SHIP>
```

and `$shipLine` refers to

```
<SHIP_LINES ValidationResult="ERROR">
…..
</SHIP_LINES>
```

Then after execution of `UPDATE_VALIDATION_RESULT`, `$ship` will refer to

```
<SHIP ValidationResult="ERROR">
…..
</SHIP>
```

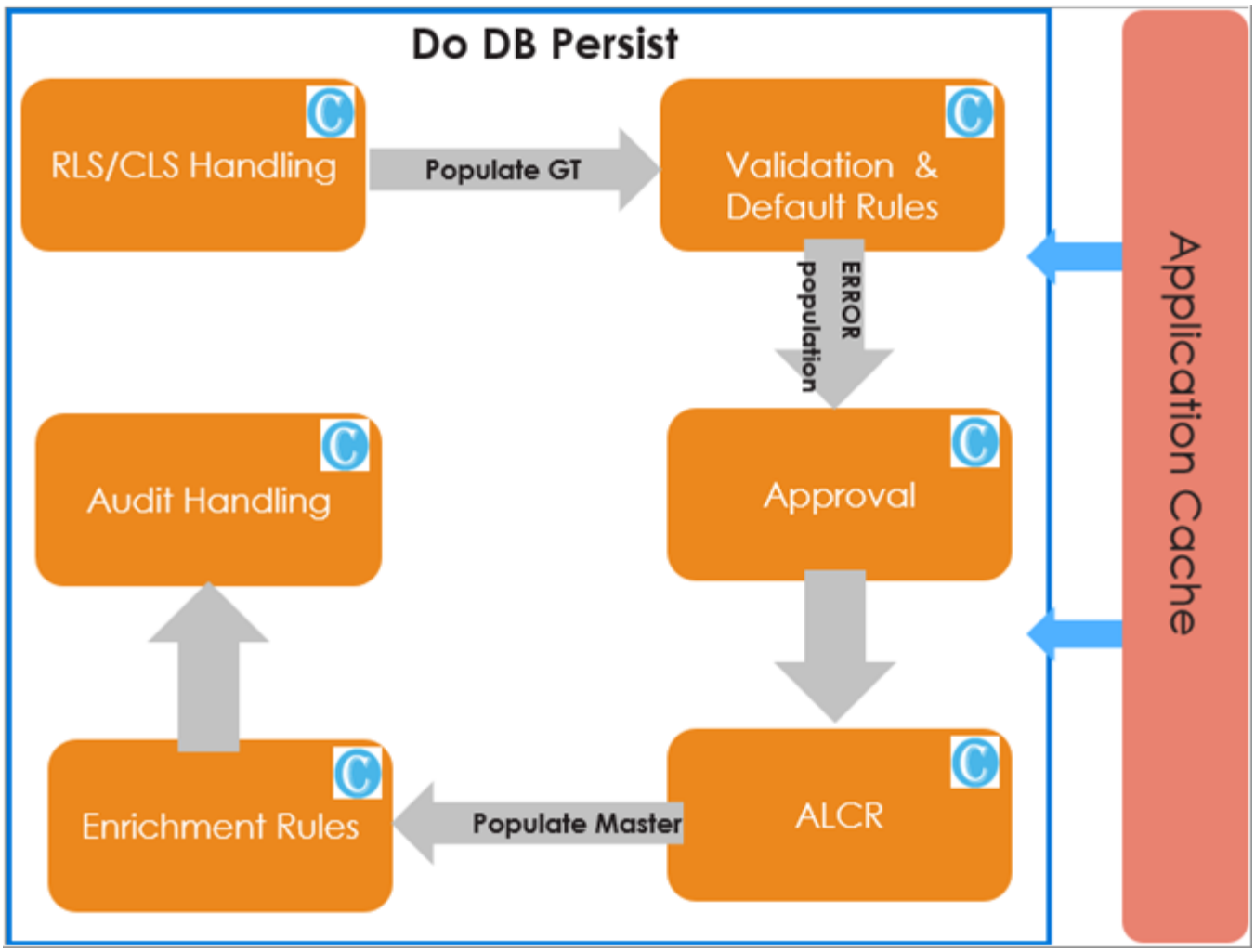
## DO_DB_PERSIST Framework Architecture

DO_DB_Persist (DDP) is a generic framework for persisting data into staging areas of MDM database objects (Tables or Views). The advantage of using this framework is that it supports custom data validation rules, custom action APIs, custom error handling APIs (in case error encountered during persist operation), raise alert based on the success or error response and approval facility on all actions (insert or update or delete or activate or deactivate) performed on underlying data.

DO_DB_Persist framework can be invoked through MDM workflows or Xrules through <DO_DB_PERSIST> X-command.

The Figure 75 displays the DO_DB_Persist framework architecture.

Figure 75: DO_DB_Persist Framework Architecture




- DDP framework architecture is componentized. The logic to handle different parts of DDP(Example: Validation, Approval, ALCR, Authorization) will be independent of each other(independent component).
- If a particular table does not have metadata related to any component (Validation, Approval, ALCR, Authorization), queries related to it will be skipped.
- All rules on a table will be cached when first time DDP is executed on the particular table and subsequent executions will read the rules from cache.
- SQLs fired into database are reduced and optimized.

Following are important points supported in Do DB Persist Framework:

- Multiple records handling in a single DO_DB_Persist request command.
- Execution of SQL based business rules as part of data validation.
- Execution on SQL based business validation rules can be skipped by setting the flag “Skip Business Rule” to “true”. This option is available when the API is invoked from backend.
- Execution of approval rules as part of data approval.

- Execution of SQL based data enrichment rules (insert, update business rules) after API processing.
- High performance while handling bulk data load leverages Teradata database inherent capabilities for optimum performance.
- Honor of data Authorization (Row Level Security and Column Level Security)
- Support of ALCR
- Support of Data Audit

**Note:**

- User ID is a mandatory attribute if DDP is invoked as an API.
- DDP does not accept Non PK columns under Document Context tag.
- DDP does not allow PK column updates.
- DDP does not update Master table cache and will drop document cache for the table.
- A parameter (ALCR_skipMasterJoinForDDP) is added in toolkit.xml to skip Master table Joins in ALCR SQLs.
- The DDP calls only those Business rules and ALCR Rules for which the columns are passed in the request.
- The DDP framework enhancements are still not implemented for following modules within the product (version DDP, Entity DDP, Excel Upload, E2E and Custom API)
- DDP API now work RowByRow way always.

SQL based validation rules can be defined through MDM UI business rule builder capability. For more details on SQL based validation rules, *refer to Chapter Business Rules in MDM Platform Server Guide.* In addition to custom validation rules, DO_DB_Persist has integral support to perform out of the box validations that check for the following database constraint compliance declared during entity creation.

- UNIQUE constraint
- UNIQUE INDEX constraint
- Referential Integrity constraint
- NOT NULL constraint
- CHECK constraint

For more details, refer *Chapter Business Rules in MDM Platform Server Guide*.

If the validations performed on input data passes, DO_DB_Persist invokes persist operation on the targeted table as per the action specified in DO_DB_Persist request. Based on the input load (number of records), DO_DB_Persist framework decides the best way to perform the persist operation.

For a single record request, default/custom APIs can be executed as part of persist operation. Default APIs perform the native database operation corresponding to specified DO_DB_Persist action without any configuration in data persist specification. However, in order to execute custom API, “api” attribute corresponding to particular action of a given document needs to be specified in data persist specification. For more details on configuring

datapersist specification, refer to *section Configuring Application Parameters in Chapter 2 Admin Tasks of Master Data Management Server Guide.*

For single record request if persist operation is executed through custom action API, SQL_Insert/SQL_Update business rules are not invoked as SQL_Insert/SQL_Update rules are executed only on successful persistence of input data (through INSERT/UPDATE/DELETE native database operation) on the given table. In case of custom action API, it remains unknown whether native operation has actually been performed on target table. To avoid data integrity loss in such scenarios, SQL_Insert/SQL_Update rules are not executed while processing single record requests through custom action API. This also implies that, if SQL_Insert/SQL_Update rule and custom API co-exist on a given document, then the operation performed through SQL_Insert/SQL_Update rule should be included within custom API definition itself.

In case of multiple record request, the persist action performed is the default native database operation corresponding to specified DO_DB_Persist action. In this scenario, even if custom action API is configured in data persist configuration specification for given document, it gets ignored. On validation failure, the input record is persisted into ERR table with error code and error severity as specified in the validation rule. After successful execution of persist action, data enrichment business rules (SQL_Insert rule and SQL_Update rule) are invoked. For more details on SQL based data enrichment rules, refer to *Chapter Business Rules in MDM Platform Server Guide.*

### Bypass SQL Based Validation Rules

You can bypass the execution of SQL based business validation rules by setting the flag "`Skip Business Rule`" to "true" as in the below example. This option is provided to reduce the processing time by bypassing the validation rules when you are confident about the quality of the data being uploaded.

Example:

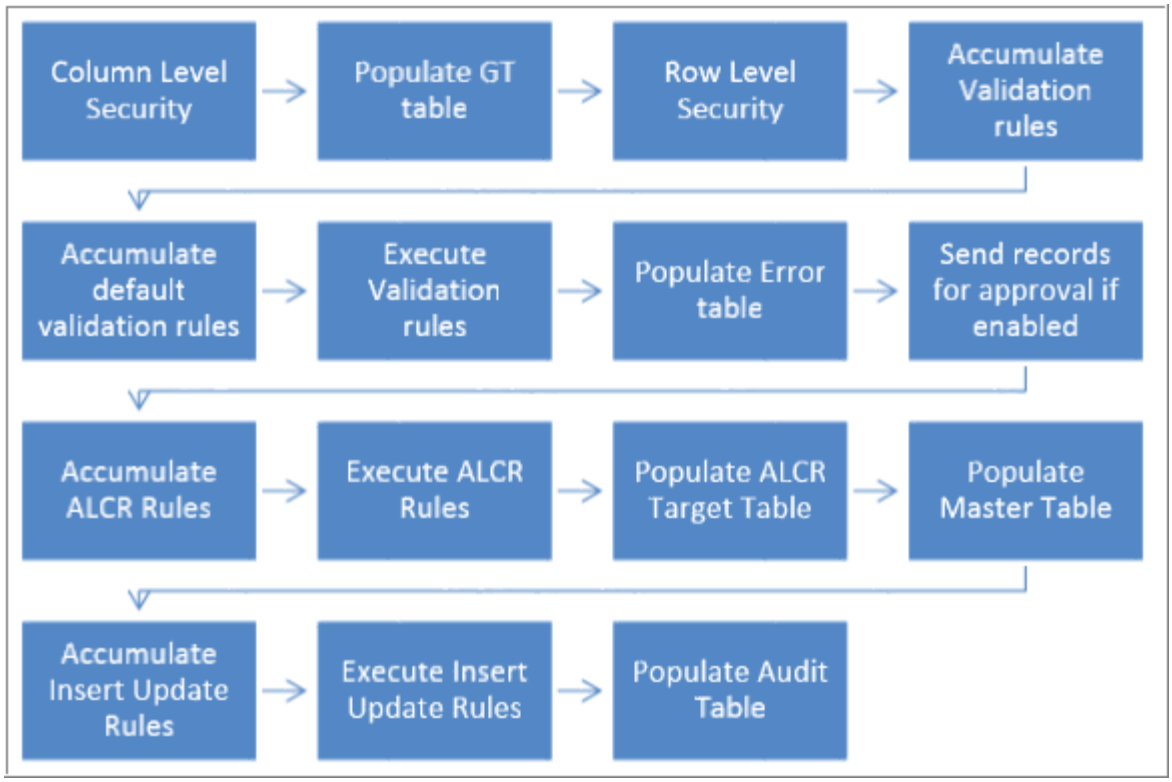
```
<REQUEST Name="doDBPersist" Source="EXTERNAL"
DocumentName="Valid_Language"  isBulkRequest="false" Action="UPDATE"
name="//BCMMasterService/doDBPersist" entireResponse="true"
UserId="USR_1"  skipBusinessRule="true" ScenarioId="0"
SysSource="UI">
  <DOCUMENT_CONTEXT>
    <Language_Id Value="56212"/>
  </DOCUMENT_CONTEXT>
  <UPDATE_PROPERTIES>
    <Description Value="21212121"/>
    <End_Date Value="12/31/9999"/>
  </UPDATE_PROPERTIES>
</REQUEST>
```

**Note:** You can set the flag to true only when the API is invoked from backend. By default, the flag will be set to false for DO_DB_Persist request invoked from UI operations.

**DO_DB_Persist Control Flow:**

The figure below displays the DO_DB_Persist process flow.

Figure 76: DO_DB_Persist Process Flow




**Syntax:**

```
<DO_DB_PERSIST Action = "VALID_ACTION" DocumentName =
"VALID_DOCUMENT_NAME" Source=”DATABASE | EXTERNAL”
MODE="ValidationOnly”

[InputServiceName=”VALID_MDM_SERVICE_NAME”]>

[ <INPUT_XML>

..........

..........

</INPUT_XML> ]

</DO_DB_PERSIST>
```

Attributes supported with DO_DB_PERSIST command includes:

- Action—type of operation to be performed on document under reference.

  Valid values: INSERT |UPDATE |DELETE | ACTIVATE | DEACTIVATE | MASS_UPDATE
- DocumentName—logical name of MDM table.
- ServiceName—target Service name of MDM.

For example: DocumentName=”Item” ServiceName=”BCM_MASTER” refers to MASTER staging table for “Item”

- Source—type of Source for input data.

  Valid Values: EXTERNAL | DATABASE (Default value is EXTERNAL)
- Mode—optional attribute, the only allowed value is “ValidationOnly”. If the mode attribute is set to "ValidationOnly", DDP Process just executes the validation business rules defined on the document and returns the status of the rules execution.
- InputServiceName—applicable only when Source is DATABASE.

  It means that input data is to be fetched from the document as mentioned in "Document" attribute value and service as specified in "InputServiceName" attribute value.

  For example, Document=”Item” and InputServiceName=”BCM_NET_CHANGE”

  It shows that input data should be fetched from NC_ITEM table.
- INPUT_XML—applicable only when Source is EXTERNAL.

  Input XML may contain property values for single record or multiple records. In case of multiple records, each record should be enclosed within <DocumentName> tags as shown in the sample requests below:

  **Sample DO_DB_PERSIST requests: Source = "EXTERNAL"**

  **Single record request**

  <docLogicalName> tag enclosing the properties is optional.

```
<DO_DB_PERSIST DocumentName="docLogicalName" Source="EXTERNAL"
Action="INSERT | UPDATE | DELETE | MASS_UPDATE | ACTIVATE |
DEACTIVATE" ServiceName="BCM_MASTER">
    [<docLogicalName>]
       <Property1_LogicalName Value="prop1Value"/>
       <Property2_LogicalName Value="prop2Value"/>
         <Property3_LogicalName Value="prop3Value"/>
         ...
    [</docLogicalName>]
</DO_DB_PERSIST>
```

  **Multiple Record Request**

  <docLogicalName> tag enclosing the properties is mandatory.

```
<DO_DB_PERSIST DocumentName="docLogicalName" Source="EXTERNAL"
Action="INSERT | UPDATE | DELETE | MASS_UPDATE | ACTIVATE |
DEACTIVATE" ServiceName="BCM_MASTER">
<docLogicalName>
<Property1_LogicalName Value="prop1Value"/>
<Property2_LogicalName Value="prop2Value"/>
        <Property3_LogicalName Value="prop3Value"/>
        ...
</docLogicalName>
<docLogicalName>
<Property1_LogicalName Value="prop1Value"/>
<Property2_LogicalName Value="prop2Value"/>
        <Property3_LogicalName Value="prop3Value"/>
```

```
        ...
</docLogicalName>
....
....
</DO_DB_PERSIST>
```

**Partial Row DO_DB_Persist:** For both single and multiple record request, partial DDP is supported. You can skip the insert or update operations on the columns using partial DDP.

**Note:** Column list for all rows in request should be same.

**Sample DO_DB_PERSIST requests: Source = "DATABASE"**

No <INPUT_XML> is required as data will be fetched from source table of the given InputServiceName.

```
<DO_DB_PERSIST DocumentName=”Item” Action=”INSERT”
AssignToVar=”thisReturn” Source=”DATABASE”
ServiceName=”BCM_MASTER” InputServiceName=”BCM_NET_CHANGE”>
</DO_DB_PERSIST>
```

Points to consider:

- InputServiceName needs to be a valid MDM staging service like BCM_INPUT_STAGING || BCM_NET_CHANGE || BCM_MASTER.
- In case of Source="DATABASE", if InputServiceName is BCM_NET_CHANGE, then SYS_NC_TYPE field of NetChange table has to be populated with any of the valid values- INSERT |UPDATE |DELETE | ACTIVATE. This scenario is applicable in case of E2E process.
- If SQL_Insert/SQL_Update rule and custom API co-exist on a given document, then the operation performed through SQL_Insert/SQL_Update rule has to be included within custom API definition itself.
- Custom API invocation is applicable only in case of single record request with Source="EXTERNAL"

# CHAPTER 10 X-Path

### What's In This Chapter

This chapter gives you information on X-Path semantics used in the MDM framework.

Topics include:


- Introduction
- X-Path Basics
- Boolean X-Path Expressions
- Node-Set Functions
- String Functions
- Boolean Functions
- Number Functions
- Arithmetic Functions
- Comparison Functions
- Time Functions
- Java's SimpleDateFormat
- Miscellaneous Functions
- Java Functions


# Introduction

The primary purpose of X-Path is to provide a means to address parts of an XML document. In addition, it makes available a number of functions for manipulating strings, numbers and Booleans. It contains a compact, non-XML syntax to facilitate use within URLs and XML attribute values. X-Path operates on the abstract, logical structure of an XML document, rather than its surface syntax. X-Path gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document.

In the MDM framework, X-Path is used as a concise way to express X-Rules. The main syntactic paradigm in X-Path is the expression. An expression is evaluated to yield an object, which has one of the following four basic types:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean (true or false)
- Number (a floating-point number)
- String (a sequence of characters)

**Note:** The X-Path functions in MDM are similar in syntax and usages to those provided under the W3C standards, but were developed to suit the needs of MDM framework and do not necessarily conform to the standards.

The X-Path functions for MDM are classified into six categories based on the type of operation:

- Node-set
- String
- Number
- Boolean
- Arithmetic
- Comparison
- Time
- Miscellaneous
- Java

Each of these classes of function is discussed in detail in the later sections of the chapter.

# X-Path Basics

The following is the general form of X-Path expression:

```
"$variableName/childNodeName[where expression]/childNodeName"
```

**Note:** '$' is essential to refer variables when X-Path is used.

The following are a few examples to understand how X-Path works.

*Example 1:*

Assume that the variable ` coDoc ' points to the following XML document:

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
```

```
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

Each of the following sub-sections contains an example of how X-Path can be used to address different parts of `coDoc`.

## Selecting a node

To select the ID node in `coDoc`,

Input:

```
<SET Var="res2" FromValue="{$coDoc/ID}"/>
<PRINT Value="{$res2}"/>
```

Output:

```
<ID Value="CO-1" />
```

```
Selecting the value of a node
```

To select the value of the `` `ID' `` node in `coDoc`:

Input:

```
<SET Var="res3" FromValue="{$coDoc/ID/@Value}"/>
```

```
<PRINT Value="{$res3}"/>
```

Output:

```
CO-1
```

## Selecting a list of nodes

To select the list of line item nodes (`ORDER_LINE_ITEM`)

Input:

```
<SET Var="res4" FromSelectList="$coDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM"/>
<PRINT Value="{$res4}"/>
```

Output:

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
```

```
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
```

If the X-Path expression in "`FromValue`" evaluates to a List then the first object in the List is returned.

Input:

```
<SET Var="res4" FromValue="{$coDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM}"/>
<PRINT Value="{$res4}"/>
```

Output:

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
```

## Selecting a list of nodes based on a condition

The following examples illustrates how you can select the list of `ORDER_LINE_ITEM` nodes where value of `GRP_MEMBER` is `no'.

*Example 1:*

Input:

```
<SET Var="result" FromSelectList ="$coDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM[GRP_MEMBER/@Value ='no']"/>
<PRINT Value="{$result}"/>
```

Output:

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
```

```
</ORDER_LINE_ITEM>

<ORDER_LINE_ITEM>

<ID Value="LI-2"/>

<CO_ID Value="CO-1"/>

<ITEM_ID Value="MONITOR"/>

<QTY Value="200"/>

<PRICE Value="69.55"/>

<GRP_MEMBER Value="no"/>

</ORDER_LINE_ITEM>
```

*Example 2:*

The following is an example to select `ORDER_LINE_ITEM` nodes where value of `ITEM_ID` is `` `MONITOR‘ ``.

Input:

```
<SET Var="res6" FromValue="{$coDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM[ITEM_ID/@Value ='MONITOR']}"/>

<PRINT Value="{$res6}"/>
```

Output:

```
<ORDER_LINE_ITEM>

<ID Value="LI-2"/>

<ITEM_ID Value="MONITOR"/>

<QTY Value="200"/>

<PRICE Value="69.55"/>

<GRP_MEMBER Value="no"/>

</ORDER_LINE_ITEM>
```

# Boolean X-Path Expressions

A Boolean X-Path Expressions is a concise way to express a condition within statement in XRules. It should be used within the test clause of an `IF_TEST` statement.

The following is the format of an X-Path expression that evaluates to Boolean is:

```
LHS Operator RHS
```

Here, LHS and RHS are operands or X-Path expressions

The Valid Operand values are:

- Null
- Number Literal, for example:10, 23.5
- String Literal, for example: 'True', 'CO-1'

Valid Operators are:

- EQUAL: '='
- NOT_EQUAL: '!='
- AND: 'and'
- OR: 'or'

The following operators work only on numbers and dates:

- GREATER: '>'
- GREATER_EQUAL: '>='
- LESS: '<'
- LESS_EQUAL: '<='

The following operators work only on numbers:

- ADD: '+'
- SUB: '-'
- MUL: 'mul'
- DIV: 'div'
- MOD: 'mod"

Valid X-Path Expression forms are:

```
"$variableName/childNodeName[where expression]/childNodeName/@Value"
```

For example, `$thisParam/RESULT/@Value`

Assume that the variable `` `coDoc' `` points to the following XML document:

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
```

```
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

*Example 1:*

The following example illustrates that, when a variable is assigned a boolean expression that evaluates to true, the value of the variable is also set to true.

Input:

```
<SET Var="res1" FromValue="{$coDoc/ID/@Value = 'CO-1'}"/>
<PRINT Value="{$res1}"/>
```

Output:

```
True
```

*Example 2:*

The following example illustrates that, when a Boolean condition in a if-test evaluates to true, the conditional statements are executed.

Input:

```
<IF_TEST Test="$coDoc/ID/@Value = 'CO-1'">
<THEN>
```

```
<PRINT Value="Found CO"/>
</THEN>
</IF_TEST>
```

Output:

```
Found CO
```

# Node-Set Functions

This section deals with the following Node-Set functions.

- count
- name

For all the examples given for the various functions, assume the variable `coDoc' points to the following sample document:

*Sample Document:*

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
```

```
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

## count

Syntax: count (X-path expression)

Return Type: number

Description: The count function is a node-set function that returns the number of nodes (or tags) in the argument.

*Example 1:*

Input:

```
<PRINTLN Value="{count($coDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM)}"/>
```

Output:

```
3
```

## name

Syntax: name (X-path expression)

Return Type: string

Description: The name function is a node set function that returns the name of the root tag of a document. In case the argument X-path expression refers to a node-set, the function returns a string that represents the expanded-name of the node, in the argument node-set, that is first in the document order.

*Example 1:*

Input:

```
<PRINTLN Value="{name($coDoc)}"/>
```

Output:

```
CUSTOMER_ORDER
```

# String Functions

This section contains the following String functions:

- string
- concat
- strlen
- starts-with
- subString
- substringBefore
- substringAfter
- normalize-space
- trim
- upperCase
- lowerCase

For all the examples given for the various functions, assume the variable `coDoc' points to the following sample document:

*Sample Document:*

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
```

```
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
<CUSTOMER_ORDER>
```

## string

Syntax: string (X-Path expression)

Return Type: string

Description: The string function converts an object to a string as follows:

A node-set is converted to a string by returning the string-value of the node in the node-set that is first in document order. If the node-set is empty, an empty string is returned.

A number is converted to a string as follows:

- NaN is converted to the string NaN
- Positive zero is converted to the string 0
- Negative zero is converted to the string 0
- Positive infinity is converted to the string Infinity
- Negative infinity is converted to the string -Infinity
- Integer: Represented in decimal form as a number with no decimal point and no leading zeros; preceded by a minus sign (-) if the number is negative.
- Decimal: Represented in decimal form as a number including a decimal point with at least one digit before the decimal point and at least one digit after the decimal point; preceded by a minus sign (-) if the number is negative.
- The boolean false value is converted to the string false. The boolean true value is converted to the string true.
- An object of a type other than the four basic types (node-set, string, number, boolean) is converted to a string in a way that is dependent on that type.

*Example 1:*

Input:

```
<PRINTLN Value="{string($coDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM[ID/
@Value = 'LI-1']/ PRICE/@Value)}"/>
```

Output:

```
103.58
```

## concat

Syntax: concat (X-Path expression, X-Path expression)

Return Type: string

Description: The concat function is a string function that returns the concatenation of its arguments.

*Example 1:*

Input:

```
<PRINTLN Value="{concat('Variable coDoc is ', name($coDoc) ,' with
ID: ',$coDoc/ID/@Value)}"/>
```

Output:

```
Variable coDoc is CUSTOMER_ORDER
```

## strlen

Syntax: strlen (X-Path expression)

Return Type: number

Description: The strlen function is a string function that returns the number of characters in the string. If the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.

*Example 1:*

Input:

```
<PRINTLN Value="{strlen(`abcde')}"/>
```

Output:

```
5
```

## starts-with

Syntax: starts-with (contains string, substring)

Return Type: boolean

Description: The starts-with function is a string function that returns true if the first argument string starts with the second argument string, otherwise returns false.

*Example 1:*

Input:

```
<PRINTLN Value="{starts-with(name($coDoc),'CO')}" />
```

Output:

```
false
```

## subString

Syntax: subString (sourceString, beginIndex, endIndex)

Return Type: string

Description: The subString function is a string function that returns a portion of the source string which is a subset of the sourceString. The subString begins at index =beginIndex and extends to the character at index =endIndex -1. The indexs are zero based, i.e. first character is index 0. The endIndex is optional argument; if not specified, then a subString beginning at beginIndex is returned.

*Example 1:*

Input:

```
<PRINTLN Value="{subString('12345',0,3)}"/>
```

Output:

```
123
```

## substringBefore

Syntax: substringBefore (sourceString, delimiter)

Return Type: string

Description: The substringBefore function is a string function that returns a portion of the source string which occurs before the delimiter.

*Example 1:*

Input:

```
<PRINTLN Value="{substringBefore('12/15/2002 12:00:00:000', ' ')}" /
>
```

Output:

```
12/15/2002
```

## substringAfter

Syntax: substringAfter (sourceString, delimiter)

Return Type: string

Description: The substringAfter function is a string function that returns a portion of the sourceString which occurs after the delimiter.

*Example 1:*

Input:

```
<PRINTLN Value="{substringAfter('12/15/2002 12:00:00:000', ' ')}" />
```

Output:

```
12:00:00:000
```

## normalize-space

Syntax: normalize-space (sourceString)

Return Type: string

Description: The normalize-space function is a string function that returns a string with leading and trailing whitespace removed. Any sequence of whitespace characters is replaced with a single space.

*Example 1:*

Input:

```
<PRINTLN Value="{normalize-space('      The    quick brown fox   ')}"
/>
```

Output:

```
The quick brown fox
```

## trim

Syntax: trim (sourceString)

Return Type: string

Description: The trim function is a string function that returns a string which is trimmed for whitespaces on the left as well as right only. The whitespaces occurring between characters are left untouched.

*Example 1:*

Input:

```
<PRINTLN Value="{trim('  The    quick brown fox  ')}" />
```

Output:

```
The quick brown fox
```

## upperCase

Syntax: upperCase (X-Path expression)

Description: The upperCase function is a string function that changes the case of the string value passed as argument to upper case; it is most useful when performing case insensitive search.

*Example 1:*

Input:

```
<PRINTLN Value="Upper Case: {upperCase('abc')}"/>
```

Output:

```
Upper Case: ABC
```

*Example 2:*

Input:

```
<SET Var="temp" FromValue="hello World"/>
<PRINTLN Value="Upper Case: {upperCase($temp)}"/>
```

Output:

```
Upper Case: HELLO WORLD
```

### lowerCase

Syntax: lowerCase (X-Path expression)

Description: The lowerCase function is a string function that changes the case of the string value passed as argument to lower case; it is most useful when performing case insensitive search.

*Example 1:*

Input:

```
<PRINTLN Value="Lower Case: {lowerCase('ABCDEFGH')}"/>
```

Output:

```
Lower Case: abcdefgh
```

*Example 2:*

Input:

```
<SET Var="temp" FromValue="HELLO WORLD"/>
<PRINTLN Value="Lower Case: {lowerCase($temp)}"/>
```

Output:

```
Lower Case: hello world
```

## Boolean Functions

This section illustrates the following Boolean functions:

- true()
- false()

### true()

Syntax: true()

Return Type: boolean

Description: The true() function is a boolean function that returns the string true.

*Example 1:*

Input:

```
<PRINTLN Value="{true()}" />
```

Output:

```
true
```

### false()

Syntax: false()

Return Type: boolean

Description: The false() function is a boolean function that returns the string false.

*Example 1:*

Input:

```
<PRINTLN Value="{false()}" />
```

Output:

```
false
```

# Number Functions

This section illustrates the following Number functions:

- int
- float
- sum

For all the examples given for the various functions, assume the variable `coDoc' points to the following sample document:

*Sample Document:*

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
<ORDER_LINE_ITEM>
```

```
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
<CUSTOMER_ORDER>
```

## int

Syntax: int (X-Path expression)

Return Type: number

Description: The int function is a number function that returns the integer value of the argument.

*Example 1:*

Input:

```
<PRINTLN Value="{int($coDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM[ID/
@Value = 'LI-1']/PRICE/@Value)}" />
```

Output:

```
103
```

### float

Syntax: float (X-Path expression)

Return Type: number

Description: The float function is a number function that returns the floating-point value of the argument.

*Example 1:*

Input:

```
<PRINTLN Value="{float($coDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM[ID/
@Value = 'LI-1']/PRICE/@Value)}" />
```

Output:

```
103.58
```

### sum

Syntax: sum (X-Path expression)

Return Type: number

Description: The sum function is a number function that returns the sum for each node in the argument X-Path expression (node-set). If the argument contains string values, then these are converted to numbers first.

*Example 1:*

Input:

```
<PRINTLN Value="{sum($coDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM/QTY/
@Value)}"/>
```

Output:

```
650.0
```

## Arithmetic Functions

This section illustrates the following Arithmetic functions:

- + (plus)
- - (minus)
- mul

- div
- mod

MDM uses the BODMAS rule for evaluating expressions containing more than one arithmetic operator.

## + (plus)

Syntax: X-Path expression1 + X-Path expression2

Description: This function performs the arithmetic addition operation.

*Example 1:*

Assuming variables: myLhs = 13, and myRhs = 3

Input:

```
<Print Value="{$myLhs + $myRhs}"/>
```

Output:

```
16
```

## - (minus)

Syntax: X-Path expression1 - X-Path expression2

Description: This function performs the arithmetic subtraction operation.

*Example 1:*

Assuming variables: myLhs = 13, and myRhs = 3

Input:

```
<Print Value="{$myLhs - $myRhs}"/>
```

Output:

```
10
```

## mul

Syntax: X-Path expression1 mul X-Path expression2

Description: This function performs the arithmetic multiplication operation.

*Example 1:*

Assuming variables: myLhs = 13, and myRhs = 3

Input:

```
<Print Value="{$myLhs mul $myRhs}"/>
```

Output:

```
39
```

### div

Syntax: X-Path expression1 div X-Path expression2

Description: This function performs the arithmetic division operation.

*Example 1:*

Assuming variables: myLhs = 13, and myRhs = 3

Input:

```
<Print Value="{$myLhs div $myRhs}"/>
```

Output:

```
4.3333335
```

### mod

Syntax: X-Path expression1 mod X-Path expression2

Description: This function performs the arithmetic modulus operation.

*Example:*

Assuming variables: myLhs = 13, and myRhs = 3

Input:

```
<Print Value="{$myLhs mod $myRhs}"/>
```

Output:

```
1
```

## Comparison Functions

This section illustrates the following comparison functions:

- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)
- = (equals)
- != (not equal)
- and
- or

### < (less than)

Syntax: X-Path expression1 < X-Path expression2

Description: This function performs the arithmetic lesser-than operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 3, and b = 13

Input:

```
<IF_TEST Test="$a &lt; $b">
<THEN>
<Print Value="lesser than works!"/>
</THEN>
</IF_TEST>
```

Output:

```
lesser than works!
```

## > (greater than)

Syntax: X-Path expression1 > X-Path expression2

Description: This function performs the arithmetic greater-than operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 3, and b = 13

Input:

```
<IF_TEST Test="$b &gt; $a">
<THEN>
<Print Value="greater than works!"/>
</THEN>
</IF_TEST>
```

Output:

```
greater than works!
```

## <= (less than or equal to)

Syntax: X-Path expression1 <= X-Path expression2

Description: This function performs the arithmetic lesser-than-or-equal-to operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 5, c = 2, and d = 3

Input:

```
<IF_TEST Test="($d + $c) &lt;= $a">
<THEN>
<PRINT Value="lesser than or equal to works!"/>
</THEN>
</IF_TEST>
```

Output:

```
Lesser than or equal to works!
```

## >= (greater than or equal to)

Syntax: X-Path expression1 >= X-Path expression2

Description: This function performs the arithmetic greater-than-or-equal-to operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 5, b = 11

Input:

```
<IF_TEST Test="$b >= ($a mul 2)">
<THEN>
<PRINT Value="greater than or equal to works!"/>
</THEN>
</IF_TEST>
```

Output:

```
greater than or equal to works!
```

## = (equals)

Syntax: X-Path expression1 = X-Path expression2

Description: This function performs the arithmetic equals operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 3, and b = 3

Input:

```
<IF_TEST Test="$a = $b">
<THEN>
<Print Value="equal works!"/>
```

```
</THEN>

</IF_TEST>
```

Output:

```
Equal works!
```

## != (not equal)

Syntax: X-Path expression1 != X-Path expression2

Description: This function performs the arithmetic not-equals operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 3, and b = 13

Input:

```
<IF_TEST Test="$a != $b">

<THEN>

<Print Value="not-equal works!"/>

</THEN>

</IF_TEST>
```

Output:

```
Not-equal works!
```

## and

Syntax: X-Path expression1 and X-Path expression2

Description: This function performs the 'and' operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 5, b = 11, c = 2, d = 8

Input:

```
<IF_TEST Test="$a &lt; $b and $d > $c">

<THEN>

<PRINT Value="And works!"/>

</THEN>

</IF_TEST>
```

Output:

```
And works!
```

### or

Syntax: X-Path expression1 != X-Path expression2

Description: This function performs the 'or' operation. It can be used in a comparison statement.

*Example:*

Assuming variables: a = 5, b = 11, c = 2, d = 8

Input:

```
<IF_TEST Test="$b &lt; $a or $d > $c">
<THEN>
<PRINT Value="Or works!"/>
</THEN>
</IF_TEST>
```

Output:

```
Or works!
```

## Time Functions

The following time related functions are available in MDM:

- date
- getYear
- getMonth
- getDay
- getHour
- getMinute
- getSecond
- duration
- incrDate
- datediff
- parseDate

### date

The following functions are used for date related operations:

- See date()
- See date(`yyyy','mon','dd','hh','MM','ss')

### date()

Syntax: date()

Return Type: string

Description: The date() function is a date function that returns the current date with current time stamp.

*Example 1:*

Input:

```
<PRINTLN Value="{date()}"/>
```

Output:

```
04/23/2002 15:12:36:983
```

### date(`yyyy','mon','dd','hh','MM','ss')

Syntax: date (year, month, day, hour, minute, second)

Return Type: string

Description: The date ('yyyy','mon','dd','hh','MM','ss') function is a date function that returns the date for the given year, month, day, hour, minute and second.

*Example 1:*

Input:

```
<PRINTLN Value="{date('2000', '02', '01')}" />
```

Output:

```
03/01/2000 00:00:00:000
```

*Example 2:*

Input:

```
<PRINTLN Value="{date('2000', '02', '01', '12', '30', '5')}" />
```

Output:

```
03/01/2000 12:30:05:000
```

## getYear

The following functions are used for year related operations:

- getYear()
- getYear (date-string)
- getYear (date-object)

### getYear()

Syntax: getYear()

Return Type: string

Description: The getYear() function is a getYear function that returns the current year.

*Example 1:*

Input:

```
<PRINTLN Value="{getYear()}"/>
```

Output:

```
2002
```

### getYear (date-string)

Syntax: getYear (date-string)

Return Type: string

Description: The getYear (date-string) function is a getYear function that returns the year of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getYear('12/12/2008')}" />
```

Output:

```
2008
```

### getYear (date-object)

Syntax: getYear(date-object)

Return Type: string

Description: The getYear(date-object) function is a getYear function that returns the year of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getYear($dateVariable)}" />
```

Output:

```
2008
```

## getMonth

The following functions are used for month related operations:

- getMonth()
- getMonth(date-string)
- getMonth(date-object)

### getMonth()

Syntax: getMonth()

Return Type: string

Description: The getMonth() function is a getMonth function that returns the current month. If the current month is April, the output would be as in example 1.

*Example 1:*

Input:

```
<PRINTLN Value="{getMonth()}"/>
```

Output:

```
3
```

### getMonth(date-string)

Syntax: getMonth(date-string)

Return Type: string

Description: The getMonth(date-string) function is a getMonth function that returns the month of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getMonth(`12/12/2008')}" />
```

Output:

```
11
```

### getMonth(date-object)

Syntax: getMonth (date-object)

Return Type: string

Description: The getMonth(date-object) function is a getMonth function that returns the month of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getMonth($dateVariable)}" />
```

Output:

```
7
```

## getDay

The following functions are used for day related operations:

- getDay()

- getDay(date-string)
- getDay(date-object)

### getDay()

Syntax: getDay()

Return Type: string

Description: The getDay() function is a getDay function that returns the current day.

*Example 1:*

Input:

```
<PRINTLN Value="{getDay()}"/>
```

Output:

```
16
```

### getDay(date-string)

Syntax: getDay(date-string)

Return Type: string

Description: The getDay(date-string) function is a getDay function that returns the day of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getDay(`12/23/2008')}" />
```

Output:

```
23
```

### getDay(date-object)

Syntax: getDay (date-object)

Return Type: string

Description: The getDay(date-object) function is a getDay function that returns the day of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getDay($dateVariable)}" />
```

Output:

```
18
```

## getHour

The following functions are used for hour related operations:

- getHour()
- getHour(date-string)
- getHour(date-object)

### getHour()

Syntax: getHour()

Return Type: string

Description: The getHour() function is a getHour function that returns the current hour

*Example 1:*

Input:

```
<PRINTLN Value="{getHour()}"/>
```

Output:

```
10
```

### getHour(date-string)

Syntax: getHour(date-string)

Return Type: string

Description: The getHour(date-string) function is a getHour function that returns the hour of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getHour(`12/12/2008 3:26:05:02')}" />
```

Output:

```
3
```

### getHour(date-object)

Syntax: getHour (date-object)

Return Type: string

Description: The getHour(date-object) function is a getHour function that returns the hour of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getHour($dateVariable)}" />
```

Output:

```
3
```

## getMinute

The following functions are used for minute related operations:

- getMinute()
- getMinute(date-string)
- getMinute(date-object)

### getMinute()

Syntax: getMinute()

Return Type: string

Description: The getMinute() function is a getMinute function that returns the current minute.

*Example 1:*

Input:

```
<PRINTLN Value="{getMinute()}"/>
```

Output:

```
10
```

### getMinute(date-string)

Syntax: getMinute (date-string)

Return Type: string

Description: The getMinute(date-string) function is a getMinute function that returns the Minute of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getMinute(`12/12/2008 3:26:05:02')}" />
```

Output:

```
05
```

### getMinute(date-object)

Syntax: getMinute (date-object)

Return Type: string

Description: The getMinute(date-object) function is a getMinute function that returns the Minute of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getMinute($dateVariable)}" />
```

Output:

```
05
```

## getSecond

The following functions are used for second related operations:

- getSecond()
- getSecond(date-string)
- getSecond(date-object)

### getSecond()

Syntax: getSecond()

Return Type: string

Description: The getSecond() function is a getSecond function that returns the current second.

*Example 1:*

Input:

```
<PRINTLN Value="{getSecond()}"/>
```

Output:

10

### getSecond(date-string)

Syntax: getSecond(date-string)

Return Type: string

Description: The getSecond(date-string) function is a getSecond function that returns the second of the date string.

*Example 1:*

Input:

```
<PRINTLN Value="{getSecond(`12/12/2008 3:26:05:02')}" />
```

Output:

```
02
```

### getSecond(date-object)

Syntax: getSecond (date-object)

Return Type: string

Description: The getSecond(date-object) function is a getSecond function that returns the second of the date object.

*Example 1:*

Input:

```
<PRINTLN Value="{getSecond($dateVariable)}" />
```

Output:

```
03
```

## duration

Syntax: duration(days, hours, minutes, seconds)

Return Type: number

Description: The duration function returns the duration in terms of seconds for the given number of days, hours, minutes and seconds.

*Example 1:*

Input:

```
<PRINTLN Value="{duration(0, 0, 1, 3)}" />
```

Output:

```
63
```

## incrDate

Syntax: incrDate(date, duration)

Return Type: string

Description: The duration function returns a date, incrementing the given date by the given duration. If a negative duration is given, then the date is decremented by that value.

*Example 1:*

Input:

```
<PRINTLNValue="{incrDate(`12/12/2002 12:00:00:000', duration(0, 0,
1, 3))}"/>
```

Output:

```
12/12/2002 12:01:03:000
```

*Example 2:*

Input:

```
<PRINTLN Value="{incrDate(`12/12/2002 12:00:00:000', -duration(0, 0,
0, (-3)))}"/>
```

Output:

```
12/12/2002 11:59:57:000
```

### datediff

Syntax: datediff(date1, date2)

Return Type: number

Description: The datediff function returns the duration in seconds by taking the difference: date1 - date2.

*Example 1:*

Input:

```
<PRINTLN Value="{dateDiff(`12/15/2002 12:00:00:000', `12/12/2002')}"
/>
```

Output:

```
302400
```

### parseDate

Syntax: parseDate(date1, format)

Return Type: string

Description: The parseDate function parses a string using a specified format and returns a datetype in the standard date format.

Currently this will support date formats as supported by java's SimpleDateFormat (See the section below for further information).

*Example 1:*

Input:

```
<SET Var="myDate" FromValue="01-Apr-2005"/>
```

```
<PRINTLN Value="Parse date using format dd-MMM-yyyy:
{parseDate($myDate,'dd-MMM-yyyy')}" />
```

Output:

```
04/01/2005 00:00:00:0007/2004
```

# Java's SimpleDateFormat

## Date and Time Patterns

Date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

Table 88: Pattern letters

| Letter | Date or Time Component | Presentation | Examples |
|---|---|---|---|
| G | Era designator | Text | AD |
| y | Year | Year | 1996,96 |
| M | Month in year | Month | July; Jul; 07 |
| w | Week in year | Number | 27 |
| W | Week in month | Number | 2 |
| d | Date in year | Number | 189 |
| D | Date in month | Number | 10 |
| F | Day of week in month | Number | 2 |
| E | Day in week | Text | Tuesday,Tue |
| a | AM/ PM marker | Text | PM |
| H | Hour in day (0-23) | Number | 0 |
| k | Hour in day (1-24) | Number | 24 |
| K | Hour in am/pm (0-11) | Number | O |
| h | Hour in am/pm (1-12) | Number | 12 |
| m | Minute in hour | Number | 30 |
| s | Second in minute | Number | 55 |
| S | Millisecond | Number | 978 |
| z | Time zone | General time zone | Pacific Standard Time; PST; GMT-08:00 |
| Z | Time zone | RFC 822 time zone | -0800 |

Pattern letters are usually repeated, as their number determines the exact presentation:

- Text: For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
- Number: For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
- Year: For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.

For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.

For parsing with the abbreviated year pattern ("y" or "yy"), SimpleDateFormat must interpret the abbreviated year relative to some century. It does this by adjusting dates to be within 80 years before and 20 years after the time the SimpleDateFormat instance is created. For example, using a pattern of "MM/dd/yy" and a SimpleDateFormat instance created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits, as defined by Character.isDigit(char), will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, "-1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

- Month: If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.
- General time zone: Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:

```
GMTOffsetTimeZone:
   GMT Sign Hours : Minutes
Sign: one of
      + -
Hours:
   Digit
       Digit Digit
Minutes:
      Digit Digit
Digit: one of
      0 1 2 3 4 5 6 7 8 9
```

- Hours must be between 0 and 23, and Minutes must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, RFC 822 time zones are also accepted.

- RFC 822 time zone: For formatting, the RFC 822 4-digit time zone format is used:

```
RFC822TimeZone:
      Sign TwoDigitHours Minutes
TwoDigitHours:
         Digit Digit
```

TwoDigitHours must be between 00 and 23. Other definitions are as for general time zones.

For parsing, general time zones are also accepted.

SimpleDateFormat also supports localized date and time pattern strings. In these strings, the pattern letters described above may be replaced with other, locale dependent, pattern letters. SimpleDateFormat does not deal with the localization of text other than the pattern letters; that's up to the client of the class.

*Examples*

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Table 89: Time Zone Pattern

| Date and Time Pattern | Result |
|---|---|
| "yyyy.MM.dd G 'at' HH:mm:ss z" | 2001.07.04 AD at 12:08:56 PDT |
| "EEE, MMM d, "yy" | Wed, Jul 4, '01 |
| "h:mm a" | 12:08 PM |
| "hh 'o"clock' a, zzzz" | 12 o'clock PM, Pacific Daylight Time |
| "K:mm a, z" | 0:08 PM, PDT |
| "yyyyy.MMMMM.dd GGG hh:mm aaa" | 02001.July.04 AD 12:08 PM |
| "EEE, d MMM yyyy HH:mm:ss Z” | Wed, 4 Jul 2001 12:08:56 -0700 |
| "yyMMddHHmmssZ" | 010704120856-0700 |

# Miscellaneous Functions

This section illustrates the following miscellaneous functions:

- getFirstDoc
- getDocument
- isValidValue
- getDBInfo
- getWorkingDirectory

For all the examples given for the various functions, assume the variable `coDoc' points to the following sample document:

Sample Document:

```
<CUSTOMER_ORDER>
<ID Value="CO-1"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<ORDER_LINE_ITEMS>
```

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

## getFirstDoc

Syntax: getFirstDoc(//ServiceName/TableName, col1, <expr1>, col2, <expr2>.....)

Return Type: XML

Description: The getFirstDoc function will fetch the first document matching the criteria from database for the specified Xservice. In the absence of the ServiceName, default service (i.e. the one from which the function is being called) will be used.

*Example 1:*

Input:

```
<PRINTLN Value="{getFirstDoc('//TEST/ORDER_LINE_ITEM', 'CO_ID', 'CO-1')}" />
```

Output:

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
```

## getDocument

Syntax: getDocument(//ServiceName/TableName, col1, <expr1>, col2, <expr2>.....)

Return Type: XML

Description: The getDocument function will fetch all the documents matching the criteria from database for the specified Xservice. In the absence of the ServiceName, default service (i.e. the one from which the function is being called) will be used.

*Example 1:*

Input:

```
<PRINTLNValue="{getDocument('ORDER_LINE_ITEM', 'CO_ID', 'CO-1')}" />
```

Output:

```
<ORDER_LINE_ITEM>
<ID Value="LI-1"/>
<CO_ID Value="CO-1"/>
<ITEM_ID Value="CPU"/>
<QTY Value="350"/>
<PRICE Value="103.58">
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-2"/>
```

```
<ITEM_ID Value="MONITOR"/>
<QTY Value="200"/>
<PRICE Value="69.55"/>
<GRP_MEMBER Value="no"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-3"/>
<ITEM_ID Value="KEYBOARD"/>
<QTY Value="100"/>
<PRICE Value="14.00"/>
<GRP_MEMBER Value="yes"/>
</ORDER_LINE_ITEM>
```

## isValidValue

Syntax: isValidValue(X-Path expression, <val1>, <val2>.....)

Return Type: Boolean

Description: The isValidValue function will return true if the node-list from the X-Path expression contains one of the listed values - val1, val2, and so on; otherwise returns false.

*Example 1:*

Input:

```
<IF_TEST Test=" isValidValue ($coDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM[ID/@Value='LI-1']/GRP_MEMBER/@Value, 'yes',
`maybe')">
<THEN>
<PRINTLN Value="Group Member"/>
</THEN>
<ELSE>
<PRINTLN Value="Invalid Group Member"/>
</ELSE>
</IF_TEST>
```

Output:

```
Invalid Group Member
```

### getDBInfo

Syntax: getDBInfo()

Return Type: XML

Description: The getDBInfo function will return the database connection information.

*Example 1:*

Input:

```
<PRINTLN Value="{getDBInfo()}"/>
```

Output:

```
<DATA_BASE_META_INFO>
<PRODUCT_NAME Value="Teradata"/>
<PRODUCT_VERSION Value="Teradata v26.1 Release …"/>
<USER_ID Value=" MDM"/>
</DATA_BASE_META_INFO>
```

### getWorkingDirectory

Syntax: getWorkingDirectory ()

Return Type: String

Description: The getWorkingDirectory function will return the server's root directory.

*Example 1:*

Input:

```
<PRINTLN Value="{ getWorkingDirectory()}"/>
```

Output:

```
C:/teradata/studio/bin
```

# Java Functions

MDM supports embedding Java functions within X-Rules to handle complex processing tasks. The following X-Path functions are used to invoke Java functions:

- nativeFunction
- instanceFunction
- instanceMethod

### nativeFunction

Syntax:

nativeFunction('myClassName', 'myMethodName', 'param1',..., 'paramN')

Where Parameters param1,..., paramN can use X-Paths expressions.

Syntax of the Class:

```
public Class myClassName
{
...................
...................
Static public DataType myMethodName (DataType1 param1 ... DataTypeN
paramN)
{
...................
...................
}
}
```

Description:

A Native Function is used to perform complex tasks in Java. It is a static method of a class and does not require instantiating the class. Native Function can be used only when the method is a not override-able; if the method is such that it may need to be overridden in specific implementations, it is better to use an Instance Function (see next section).

Input:

```
<SET Var="myId" FromValue="{nativeFunction('com.teradata.X-
Core.utils.UniqueIdGenerator', 'getUniqueId', 'PO')}"/>
<PRINT Value="{$myId}" />
```

Output:

```
PO-78691
```

## instanceFunction

Syntax:

instanceFunction('myClassName', 'myMethodName', 'param1',..., 'paramN')

Where Parameters param1,..., paramN can use X-Paths expressions.

Syntax of the Class:

```
public class myClassName
{
// A public no-argument default constructor is required
```

```
public myClassName()

{

}

.................................................

.................................................

public DataType myMethodName ( DataType1 param1,..., DataTypeN paramN
)


.................................................

.................................................

}

}
```

Description:

An Instance function is used to perform complex tasks in Java. Instance function is similar to Native function except that the function is not 'static' and (hence) the mechanism used to make the invocation is different. MDM instantiates an object of the specified class name the first time a method is invoked on it in the transaction. The instantiation happens using a public no-argument default constructor the class is required to provide. It then invokes the function. The object is re-used for all the later invocations during the transaction.

There are 2 added benefits of instance functions not available in native functions.

1. Instance functions allow the functions to update some state (instance) variables and re-use the state across method invocations within a single transaction.
2. There is a class mapping facility. To understand this feature, consider that some rules use instance functions on say class `A'. In a customer deployment, some of the methods in class `A' are needed to behave differently. Another class `B' can be written extending class `A' that overrides these methods. In the class mapping setup, A may be mapped to `B'. During execution, class `B' will be instantiated wherever `A' is mentioned and the methods are invoked on `B'. If some methods don't need to change, class `B' will automatically inherit them.

*Example 1:*

Input:

```
<SET Var="myId" FromValue="{instanceFunction('com.teradata.X-
Core.utils.UniqueIdGenerator', 'getUniqueId', 'PO')}"/>

<PRINT Value="{$myId}"/>
```

Output:

```
PO-78654389
```

*Example2:*

Setting up Class-Mapping

Add the following tag to the service's .xml configuration file:

```
<class-mapper File="myclassmapping.xml"/>
```

The file classmapping.xml will have content similar to the following:

```
<class-mappings>

<class Name="com.teradata.xservice.origpackage.OrignalClass"
MapTo="com.teradata.xservice.newpackage.NewClass"/>

</class-mappings>
```

This mapping will cause the class com.teradata.xser.newpackage.NewClass to be used in the X-Rules wherever class com.teradata.xservice.origpackage.OriginalClass is referred to in an `INSTANCE_FUNCTION` statement.

## instanceMethod

Syntax:

instanceMethod($myObjVar, 'myMethodName', 'param1',..., 'paramN')

Where:

- myMethodName: Method to invoke,
- myObjVar: Object to invoke the method on (can use X-Path expression)
- param1,..., paramN: Parameters (can use X-Paths expressions)

Description:

Instance method statement is yet another way to invoke a method on a Java object. The difference between instance method and instance function is that instance method allows invocation of a method on a Java object whose reference is already available within the context of a rule.

*Example1:*

First, get a handle to the customer order instance using the ID:

```
<SET Var="coObj"
FromValue="{nativeFunction('com.teradata.xservice.orderadmin.sce.ord
er.reg.OrderEntityFactory', 'getOrderEntity', '{$thisParam/ID/
@Value}',' CUSTOMER_ORDER',' {$thisContext}')}"/>
```

Now, to invoke a method on the customer object instance:

Input:

```
<SET Var="invoiceValue" FromValue="{instanceMethod($coObj,
'getTotalEstimatedInvoiceValue')}"/>

<PRINTLN Value="The estimated total invoice value is: {$invoiceValue}
"/>
```

Output:

```
The estimated total invoice value is 993.45
```

# CHAPTER 11 MDM JSON Framework

### What's In This Chapter

This chapter gives you information on MDM JSON framework.

Topics include:


- Introduction
- Standards for JSON Framework
- XML and Equivalent JSON Request and Response


# Introduction

The MDM REST framework allows the users to access the MDM solution using REST based web service calls. The framework supports the JSON data format for both request and response APIs. Using this framework, the user can invoke MDM X-Rules and also perform CRUD operations on the tables defined in MDM.

# Standards for JSON Framework

The below section describes the JSON framework standards for X-rules and CRUD APIs on MDM based tables and document views.


- X-Rules
- CRUD APIs


## X-Rules

Any X-Rule defined with public access modifier can be invoked using the REST framework. Example: For a rule named "getOperations" defined in service "PLATFORM", the REST endpoint would be:

```
$MDM_BASE_URL/api/v1/services/PLATFORM/rules/getOperations
```

To invoke the X-Rule, the user has to perform a POST request on the above REST end point. Any input parameters defined for the X-Rule need to be passed in the request body as below:

The original request to the X-Rule willbe in XML format as shown below:

<REQUEST Name="getOperations" ServiceName="PLATFORM">

<UserId Value = "DepMgr"/>

<Type Value = "All"/>

</REQUEST>

To invoke this rule the equivalent JSON input for the body of the request will be as below:

{

"UserId" : "DepMgr",

"Type" : "All"

}

The endpoint would be $MDM_BASE_URL/api/v1/services/PLATFORM/rules/getOperations

## CRUD APIs

Any tables defined in MDM can be accessed by the MDM user using the REST framework. User can perform query, create, update and delete operations on the tables.

The following section describes the Create Read Update and Delete (CRUD) APIs in MDM. In RESTful APIs, to create a new record, use “POST” request, to read a record, use “GET” request, to update a record, use “PUT” request and to delete a record, use “DELETE” request.

For performing CRUD APIs in MDM, set the parameter REST_WEB_SERVICE_ENABLED="true" in both xerver.xml and xserverweb.xml available at: *<MDM_Install_Directory>\cfg\properties*

### Create API

To perform create operation on given table, use POST request as below:

#### Sample POST Request

```
POST /mdm/api/v1/services/{Service Name}/tables/{Table Name}
```

#### Sample POST Request Body

```
{
    "id": "3736",
    "name": "Simon Patrick Moon",
    "gender": "M",
    "addrLine": "Chausseestrasse 9017 Bavaria Hinterschmiding 94146",
    "city": "Hinterschmiding",
    "state": "Bavaria",
    "zipCode": "94146",
    "email": "simon.moon@illjord.com"
```

```
}
```

### Responses of POST Request

The below table displays the response code, status and the response description for the Create API request.

Table 90: Responses for POST API

| Code | Status | Description |
|---|---|---|
| 201 | Ok | Record created successfully. |
| 400 | Bad Request | Malformed syntax or missing/incorrect parameters. |
| 404 | Not Found | Details missing in the database. |
| 401 | Unauthorized | Invalid/missing/expired token.. |
| 403 | Forbidden | Missing customer 360 permission |
| 500 | Internal Server Error | Internal server error. |
| 409 | Conflict | Conflict (duplicate PK, null constraints) |

### Sample POST Request Response Header

The response will contain resource-location and resource-ID in the response header as below.

```
Location : {base path}/mdm/api/v1/services/BCM_MASTER/tables/
Composite_pk_table?id=3736
```

```
X-Resource-Id : id=3736
```

In case of composite PK, response header will have the PK information as below:

```
Location : {base path}/mdm/api/v1/services/BCM_MASTER/tables/
Composite_pk_table?id1=3736 & id2=312
```

```
X-Resource-Id : id1=3736,id2=312
```

### Sample POST Request Response in JSON

The response of the request will be in JSON format as per Teradata TIO (Technology Innovation Office) REST API standard. The response of the create record will have entire records created as response.

```
{
    "id": "3736",
    "name": "Simon Patrick Moon",
    "gender": "M",
```

```
    "addrLine": "Chausseestrasse 9017 Bavaria Hinterschmiding 94146",
    "city": "Hinterschmiding",
    "state": "Bavaria",
    "zipCode": 94146,
    "email": "simon.moon@illjord.com"
}
```

## Read API

To perform read operation on give table or document view, use GET request as below:

### Sample GET Request

```
GET /mdm/api/v1/services/{Service Name}/tables/{Table Name}
```

The following two types of requests are supported to read data:

- Non Paging Request

  For the non-paging request, user need not pass any query-param related to pagination as in the below sample. You need to call the below URL with appropriate values for service name and table name.

  ```
  GET /mdm/api/v1/services/{Service Name}/tables/{Table Name}
  ```

- Paging Request

  For paging request, user need to pass query-param related to pagination like page and page-size as in the below sample. You need to call above URL with appropriate values for service and table name.

  ```
  GET /mdm/api/v1/services/{Service Name}/tables/{Table Name}?page=value&page-size=value
  ```

  The following points are applicable for GET request:

  - The page parameter must be passed for paginated request, else it will be considered as non-paging request.
  - The page parameter value is the page number of a particular page, which the user wants to read, it starts from 1.
  - The page must be greater than or equal 0 (zero), else the request will be considered as non-paging request.
  - If the page parameter value is passed as “0”, it will be considered as first page only, that is page =1.
  - The page-size parameter value is the number of records in the current page, if this value is not passed, default page size of MDM will be taken from the below file.

    ```
    file : \platform_services\web\resources\platformclient\WEB-INF\core\model\defaults.cnd
    property : <util:set-system-property name="com.teradata.platform.web.x2.util.configurability.maxRows" value="10"/>
    ```

  - Space must not be included in query parameter.

For both paging and non paging requests, you can also pass filter and sort parameters in request as described below.

- Filtering Read Result

  To filter the result set, you need to pass query parameter in the URL as below:

  ```
  GET mdm/api/v1/services/{Service Name}/tables/{Table Name}? colName1=value&colName2=value
  ```

  URL: {base url}/mdm/api/v1/services/{Service Name}/tables/{Table Name}

  query param: colName1=value&colName2=value (passing multiple values with separated by '&')

  **Note:** Only equal ("=") operator is supported and space must not be provided in query parameter.

- Sorting Read Result Set

  To sort the result set, you need to pass query parameter in the URL as below:

  ```
  GET /mdm/api/v1/services/{Service Name}/tables/{Table Name}? sort = colName1:ASC,colName2:DESC
  ```

  URL: {base url}/mdm/api/v1/services/{Service Name}/tables/{Table Name}

  query param: sort=colName1:ASC,colName2:DESC

  **Note:**

  If no value is passed for sort parameter, the default sort order will be Ascending order.

  Passing multiple values with separated by ' , '. The column name and order type separated by ' : ' and space must not be provided in query parameter.

### Responses of GET Request

The below table displays the response code, status and the response description for the Get API request.

Table 91: Responses for GET API Request

| Code | Status | Description |
|---|---|---|
| 200 | Ok | Got list of table data |
| 400 | Bad Request | Malformed syntax or missing/incorrect parameters. |
| 404 | Not Found | Details missing in the database. |
| 401 | Unauthorized | Invalid/missing/expired token.. |
| 403 | Forbidden | Missing customer 360 permission |
| 500 | Internal Server Error | Internal server error. |

### Sample GET Response Header

In case of paging request, the response header will contain the X-Page Size, X-Page, X-Total Count and X-Page-Last details as in the below sample response header.

In case of non-paging request, the response header will contain only X-Total-count.

#### Sample GET Response in JSON

The response of the request will be in JSON format as per TIO standard.

```
[
   {
    "id": 3736,
    "name": "Simon Patrick Moon",
    "gender": "M",
    "addrLine": "Chausseestrasse 9017 Bavaria Hinterschmiding 94146",
    "city": "Hinterschmiding",
    "state": "Bavaria",
    "zipCode": 94146,
    "email": "simon.moon@illjord.com"
   },
   {
    "id": 1610,
    "name": "Caroline Ana Faria",
    "gender": "F",
    "addrLine": "9999 Alexander Valley Rd. TX Amarillo 79123",
    "city": "Amarillo",
    "state": "TX",
    "zipCode": 79123,
    "email": "caroline.faria@teratitan.com"
   }
]
```

## Update API

To perform update operation on give table or document view, use PUT request as below:

#### Sample PUT Request

```
PUT /mdm/api/v1/services/{Service Name}/tables/{Table Name}
```

#### Sample PUT Request Body

```
{
    "id": "3736",
    "zipCode": "12345"
```

```
}
```

In the request body, pass the primary key column with the updating property in JSON object form. The request body will be internally divided into two parts:

- Primary key columns: it will contain the primary key column present in the input body to filter the particular records.
- Updating columns: it will contain the remaining columns of the input body, which will be updated with the input values and extra columns will be ignored (which is not a part of table)

### Responses of PUT Request

The below table displays the response code, status and the response description for the PUT API request.

Table 92: Responses for PUT API

| Code | Status | Description |
|---|---|---|
| 200 | OK | Record updated successfully. |
| 400 | Bad Request | Malformed syntax or missing/incorrect parameters. |
| 404 | Not Found | Details missing in the database. |
| 401 | Unauthorized | Invalid/missing/expired token.. |
| 403 | Forbidden | Missing customer 360 permission |
| 500 | Internal Server Error | Internal server error. |
| 409 | Conflict | Conflict (duplicate PK, null constraints) |

### Sample PUT Request Response in JSON

The response of the request will be in JSON format as per Teradata TIO standard.

```
[
    {
     "id": 3736,
     "name": "Simon Patrick Moon",
    "gender": "M",
    "addrLine": "Chausseestrasse 9017 Bavaria Hinterschmiding 94146",
    "city": "Hinterschmiding",
    "state": "Bavaria",
    "zipCode": "12345",
    "email": "simon.moon@illjord.com"
    }
```

```
]
```

### Delete API

To perform delete operation on give table or document view, use DELETE request as below:

#### Sample DELETE Request

```
DELETE /mdm/api/v1/services/{Service Name}/tables/{Table Name}
```

#### Sample DELETE Request Body

```
{
    "id": "3736",
}
```

In the DELETE request body, pass any column name and value (not necessarily PK) in JSON format. If no body parameters are passed, delete will not be triggered and the operation will fail with Bad request error.

#### Responses of DELETE Request

The below table displays the response code, status and the response description for the DELETE API request.

Table 93: Responses for Delete API

| Code | Status | Description |
|---|---|---|
| 204 | OK | Record deleted successfully. |
| 400 | Bad Request | Malformed syntax or missing/incorrect parameters. |
| 404 | Not Found | Details missing in the database. |
| 401 | Unauthorized | Invalid/missing/expired token.. |
| 403 | Forbidden | Missing customer 360 permission |
| 500 | Internal Server Error | Internal server error. |
| 409 | Conflict | Conflict (duplicate PK, null constraints) |

# XML and Equivalent JSON Request and Response

Table 94 lists the JSON input format for invoking X-Rules through REST API endpoints, example: *API Endpoint (POST) http://$MDM_BASE_URL/api/v1/services/{$ServiceName}/rules/{$RuleName}*

Table 94: XML and Equivalent JSON Request

| XML Request | JSON Input Format (Request Body) |
|---|---|
| <REQUEST Name=" getMatchingProfiles"><br><Entity Value="Tables"/><br><NonMDMTables Value="Yes"/><br></REQUEST> | {<br>"Entity" : "Tables",<br>"NonMDMTables": "Yes"<br>} |
| <REQUEST Name=" getMatchingProfiles "><br><CustomTables Value="No"/><br><Service Value="BCM_MASTER"/><br><Service Value="MDMServices"/><br></REQUEST> | {<br>"CustomTables" : "No",<br>"Service" :["BCM_MASTER","MDMServices"]<br>} |
| <REQUEST Name=" getMatchingProfiles"><br><CustomTables Value="No"/><br><Tables><br><Table Value="Code_Set"/><br></Tables><br></REQUEST> | {<br>"CustomTables" : "No",<br>"Tables": {<br>"Table": "Code_Set"<br>}<br>} |
| <REQUEST Name=" getMatchingProfiles"><br><CustomTables Value="No"/><br><Tables><br><Table Value="Code_Set"/><br><Table Value="AttributeSet"/><br></Tables><br></REQUEST> | {<br>"CustomTables" : "No",<br>"Tables": {<br>"Table": ["Code_Set","AttributeSet"]<br>}<br>} |
| <REQUEST Name=" getMatchingProfiles "><br><CustomTables Value="No"/><br><Services><br><Service Value="Code_Set"/><br><Service Value="AttributeSet"/><br></Services><br><Tables><br><Table Value="Code_Set"/><br><Table Value="AttributeSet"/><br></Tables><br></REQUEST> | {<br>"CustomTables" : "No",<br>"Services":{<br>"Service": ["Code_Set","AttributeSet"]<br>},<br>"Tables": {<br>"Table": ["Code_Set","AttributeSet"]<br>}<br>} |

Table 94: XML and Equivalent JSON Request

| XML Request | JSON Input Format (Request Body) |
|---|---|
| <REQUEST Name="getRecords"><br><NameProp Value="Dere"<br>MatchBy="StartsWith"/><br></REQUEST> | Not Supported (Attributes Other than Value) |
| (For above use case, Rule input can be defined as below)<br><br><REQUEST Name="getRecords"><br><NameProp Value="Dere"><br><MatchBy Value="StartsWith"/><br></NameProp><br></REQUEST> | {<br>"NameProp": {<br>"Value": "Dere",<br>"MatchBy": "StartsWith"<br>}<br>} |
| To Pass Pagination and Sort parameters<br><br>URL: http://$MDM_BASE_URL/api/v1/services/{$ServiceName}/rules/{$RuleName}?page={$currentpageNum}&page-size={$pageSize}&sort={$sortPropName:ASC\|DESC} | |
| <REQUEST Name="ruleName"<br>StartAtRow="pageSize*currentPageNum"<br>MaxRows="pageSize"><br><DocName Value="Code_Set"/><br><DocSrvcName Value="BCM_MASTER"/><br><orderByXml><br><sortPropName Sort="Ascending"/><br></orderByXml><br></REQUEST> | {<br>"DocName" : "Code_Set",<br>"DocSrvcName" : "BCM_MASTER"<br>}<br>Response should have attribute named "TotalRowCount" in the response header tag<br>Ex: <RESPONSE TotalRowCount="10">…</RESPONSE> |

Table 95 lists the JSON format for responses returned by the X-Rules. The header <RESPONSE> tag is ignored while converting to JSON, so any attributes of the Response header will not be included in the converted JSON value.

Table 95: X-Rule (XML) and Equivalent JSON Response

| X-Rule Response (XML Form) | JSON Response |
|---|---|
| <RESPONSE Status="Success"><br><Employee ><br><Id Value="1"/><br><Name Value="Sam"/><br></Employee><br></RESPONSE> | {<br>"Employee": {<br>"Id": "1",<br>"Name": "Sam"<br>}<br>} |
| If an element has any other attribute than “Value” or children elements, then it will be serialized as an object or otherwise it will be serialized as simple name/value pair attribute in JSON. | |
| <RESPONSE Status="Success"><br><Employee ><br><Id Value="1"/><br><Name Value="Sam"<br>Description="Employee Name"/><br></Employee><br></RESPONSE> | {<br>"Employee": {<br>"Id": "1",<br>"Name": {<br>"Description": "Employee Name",<br>"Value": "Sam"<br>}<br>}<br>} |
| If an element has both child elements and attributes | |
| <RESPONSE Status="Success"><br><Employee ><br><Id Value="1"/><br><Name Description="Employee Name"><br><FirstName Value="Sam"/><br><LastName Value="Peter"/><br></Name><br></Employee><br></RESPONSE> | {<br>"Employee": {<br>"Id": "1",<br>"Name": {<br>"Description": "Employee Name",<br>"LastName": "Peter",<br>"FirstName": "Sam"<br>}<br>}<br>} |
| Multiple elements having same name present at the same level will be grouped as array in JSON response | |

Table 95: X-Rule (XML) and Equivalent JSON Response

| X-Rule Response (XML Form) | JSON Response |
|---|---|
| <RESPONSE Status="Success"><br><Employees><br><Employee ><br><Id Value="1"/><br><Name Value="Sam"/><br></Employee><br><Employee><br><Id Value="2"/><br><Name Value="Dan"/><br></Employee><br><Employee ><br><Id Value="3"/><br><Name Value="Meg"/><br></Employee><br></Employees><br></RESPONSE> | {<br>"Employees": {<br>"Employee": [<br>{<br>"Id": "1",<br>"Name": "Sam"<br>},<br>{<br>"Id": "2",<br>"Name": "Dan"<br>},<br>{<br>"Id": "3",<br>"Name": "Meg"<br>}<br>]<br>}<br>} |
| If all the first level children of root response element are of same name and the first level child count is more than one, then the return type would be of JSON Array or otherwise it will be simple JSON Object | |
| <RESPONSE Status="Success"><br><Employee><br><Id Value="1"/><br><Name Value="Sam"/><br></Employee><br><Employee ><br><Id Value="2"/><br><Name Value="Dan"/><br></Employee><br><Employee ><br><Id Value="3"/><br><Name Value="Meg"/><br></Employee><br></RESPONSE> | [<br>{<br>"Id": "1",<br>"Name": "Sam"<br>},<br>{<br>"Id": "2",<br>"Name": "Dan"<br>},<br>{<br>"Id": "3",<br>"Name": "Meg"<br>}<br>] |
| If the user needs the response to return always as JSON array even if the first level children count is less than 1, then the user must pass request header attribute “return-response-array” as “true” in the HTTP request | |

Table 95: X-Rule (XML) and Equivalent JSON Response

| X-Rule Response (XML Form) | JSON Response |
|---|---|
| <RESPONSE Status="Success"><br><Employee ><br><Id Value="1"/><br><Name Value="Sam Peter" Description="Employee Name"><br><FirstName Value="Same"/><br><FirstName Value=""/><br></Name><br></Employee><br></RESPONSE> | [<br>{<br>"Id": "1",<br>"Name": {<br>"Description": "Employee Name",<br>"Value": "Sam Peter",<br>"FirstName": [<br>{<br>"FirstName": "Same"<br>},<br>{<br>"FirstName": ""<br>}<br>]<br>}<br>}<br>] |

## Guidelines

The below list provides the guidelines for implementing X-rules.

- It is recommended to use CRUD REST APIs instead of writing xrules for simple CRUD operations.
- Use DocumentView for compound entities.
- Use only “Value” attribute in Request/Response XML.
- Write simple rules to perform one single task or fetch one single type of data at a time.
- Do not use single rule to fetch multiple unrelated data, refactor into multiple rules. This will result in better design of the overall rules API and leads to writing reusable generic APIs.

# CHAPTER 12 PGL UI Material Design Style

### What's In This Chapter

This appendix provides details on PGL UI Material design styling guidelines.

Topics include:


- Introduction
- PGL UI Style Improvement
- New PGL Version Introduced for Material Design
- Data Grid Improvements
- Forms Improvements
- General Guidelines


# Introduction

From MDM 4.6 onwards, the styles of MDM PGL UIs are upgraded to Material Design style. The look and feel of MDM UI key components like Data Grid, Forms, Entry fields, Text Area, icons and dropdowns are updated with the new material UI styling. In addition to the styling changes, some of the UI component usability are enhanced for better user experience and capability. Implementing the new styling will change the look and feel of the PGL UI similar to the covalent angular UI.

# PGL UI Style Improvement

Use the below PGL guidelines to enable new PGL UI Material Design styles and enhance MDM UI look and feel:

## New PGL Version Introduced for Material Design

New PGL version introduced for material styles are coming up with new version (version 4) as in the below code snippet. From 4.6 onwards, if you are adding a new PGL, then by default, the version 4 will be taken and all the new styles will be applied. New changes includes significant amount of style changes and minimal amount of HTML structure changes. Any new custom workflow to be rendered in the new version has to undergo minimal amount of Java script changes, which can be modified by the developer deploying the custom PGL workflows. If you do not want to utilize the new styles and would like to render in older version, then set the version to 3.

```
<ui:page-group xmlns:ui=”http://www.teradata.com/
ui”version=’4’layout=”focused”>
```

## Data Grid Improvements

Data Grid is enhanced with modern look and feel for better user experience with the below material styling implementation.

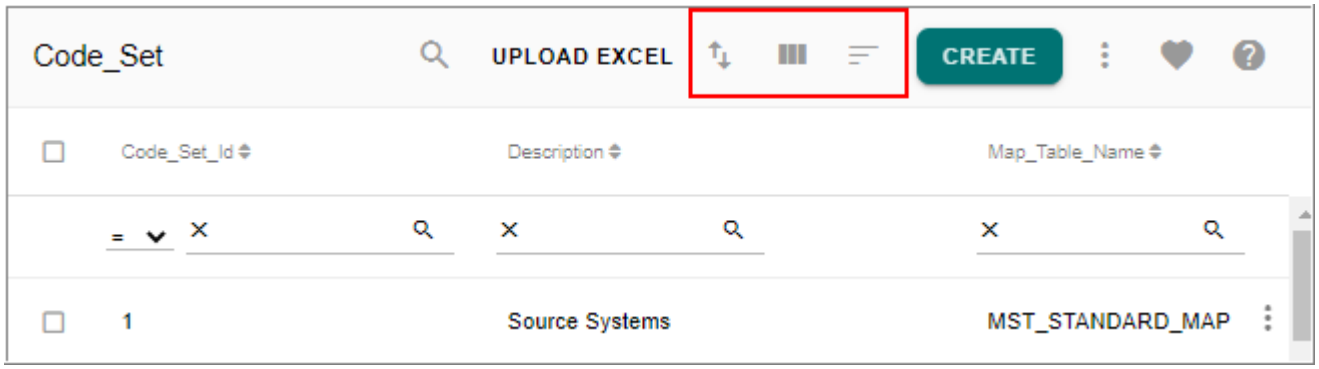
- **Angular Material Icon Support**

  By default all the icon on the MDM web UI are updated with corresponding Angular material icons as in Figure 77. To add new Angular material icons, in the <ui:image-link-field>, add the src attribute value as below.

  ```
  <ui:image-link-field src="pgl-icon mat-icon save">
  ```

  Here 'save' is material icon name.

Figure 77: Sample UI for Angular Material Icons on Toolbar



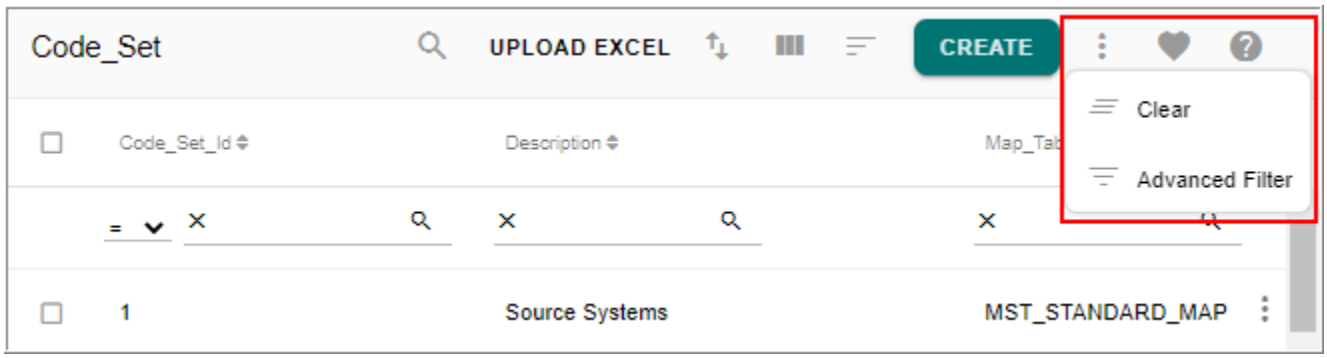

- **Toolbar Changes**
  - **Add Images Menu on Toolbar**

    To add image menu on toolbar as in Figure 78, add <image-link-field> with menuItem="true" inside <ui:header> as in below sample code.

    Example:

    ```
     <ui:image-link-field displayText="Clear" menuItem="true"
    src="pgl-icon mat-icon clear_all" title="Clear"
    onClick="javascript:onClear();" id="Clear"
    displayCondition="{$root/HEADER_BUTTON_CONDITION/displayClearBtn/
    @Value}"/>
          <ui:image-link-field displayText="Advanced Filter"
    menuItem="true" src="pgl-icon mat-icon filter_list"
    title="Advanced Filter" onClick="javascript:onFilter();"
    id="filterGif" displayCondition="{$root/DATA/showAdvancefilter/
    @Value}"/>
    ```

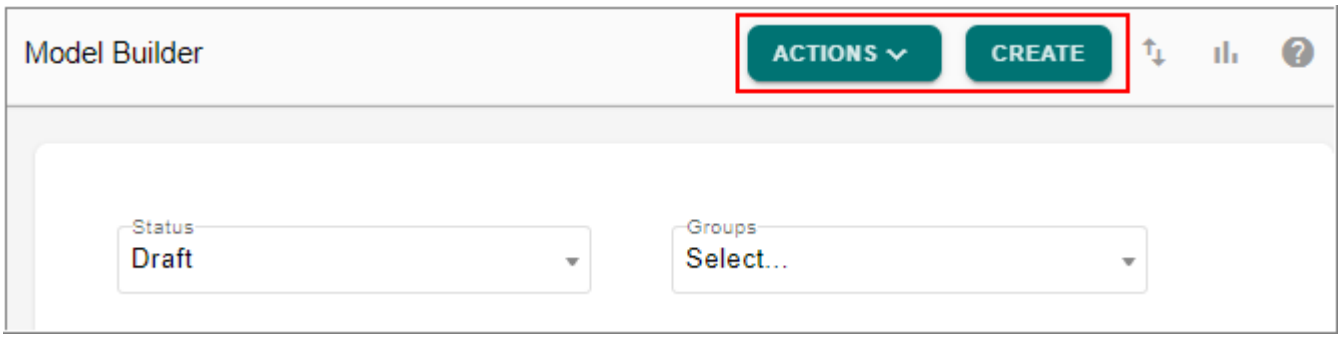
Figure 78: Sample UI for Image Menu on Toolbar




- **Buttons and Button Menus on Toolbar**

  Buttons and Button menu can be created on toolbar as in Figure 79.

Figure 79: Sample UI for Button and Button Menu on Toolbar




Add the <ui:button> in <ui:header> and <ui:table> as in below code snippet. Note that <ui:button> should be replicated at two places <ui:header> and <ui:table>

Mention headerButton="true" for buttons inside <ui:table>

Mention tableButton="true" for buttons inside <ui:header>

```
<ui:header>
        <ui:button id="upload" name="upload" displayText="Upload
Excel"/>
       <ui:buttons id="buttons" name="buttons">
         <ui:button id="addNewUserGroup" tableButton="true"
name="addNewUserGroup" displayText="Create User Group"/>
        <ui:button id="addNewUserGroupType" tableButton="true"
name="addNewUserGroupType" displayText="Manage User Group Type"/>
        <ui:button id="manageOrganizations" tableButton="true"
name="manageOrganizations" displayText="Manage Organizations"/>
</ui:buttons>
</ui:header>


 <ui:table id="searchTable" displayText="Search Results"
select="$root/USER_GROUP_PROFILES" selectable="single"
selectedId="ID" postRowIdAs="SELECTED_ID" displayTextNoRecords="no
Record Found"

      rowId="ID" scrollable="true" doPaging="true" maxRows="$root/
maxRows/@Value" totalRowCount="$root/TotalRowCount/@Value"
startAtRow="$root/StartRowCount/@Value" >
       .....
```

```
        .....
      <ui:buttons id="buttons" name="buttons">
       <ui:button id="addNewUserGroup" headerButton="true"
name="addNewUserGroup" displayText="Create User Group"
emphasized="false"/>
       <ui:button id="addNewUserGroupType" headerButton="true"
name="addNewUserGroupType" displayText="Manage User Group Type"/>
       <ui:button id="manageOrganizations" headerButton="true"
name="manageOrganizations" displayText="Manage Organizations"/>
      <ui:button id="upload" headerButton="true" name="upload"
displayText="Upload Excel"/>
</ui:buttons>
   </ui:table>
```

- **On Demand Search (Toggle Filter)**

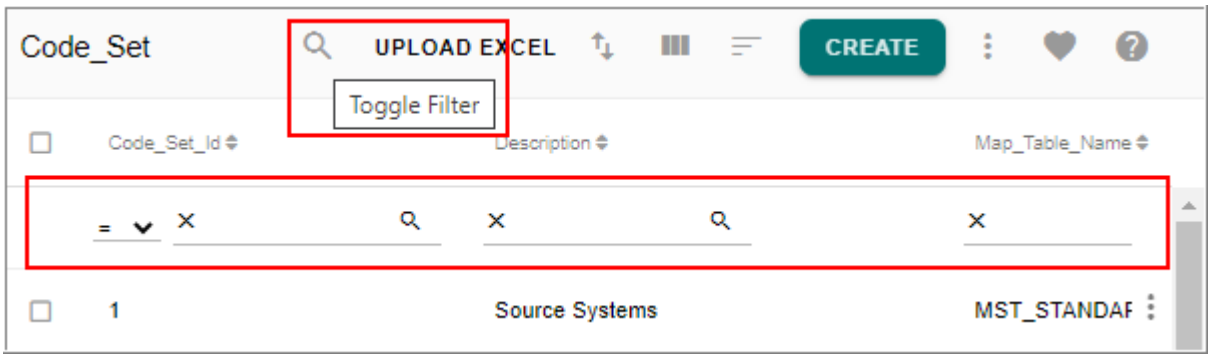
  By default, search is not enabled on UI. Toggle filter can be enabled for data table using filterToggle="true" in <ui:table> as below. When you click on toggle filter, the search and clear filter will be displayed as in Figure 80.

  ```
  <ui:table filterToggle="true">
  ```

  Also add below icon under <ui:header>

  ```
  <ui:image-link-field name="filterToggle" src="pgl-icon mat-icon
  search" title="Toggle Filter" onClick="javascript:toggleSearch();"
  id="filterToggle" />
  ```

Figure 80: Toggle Filter



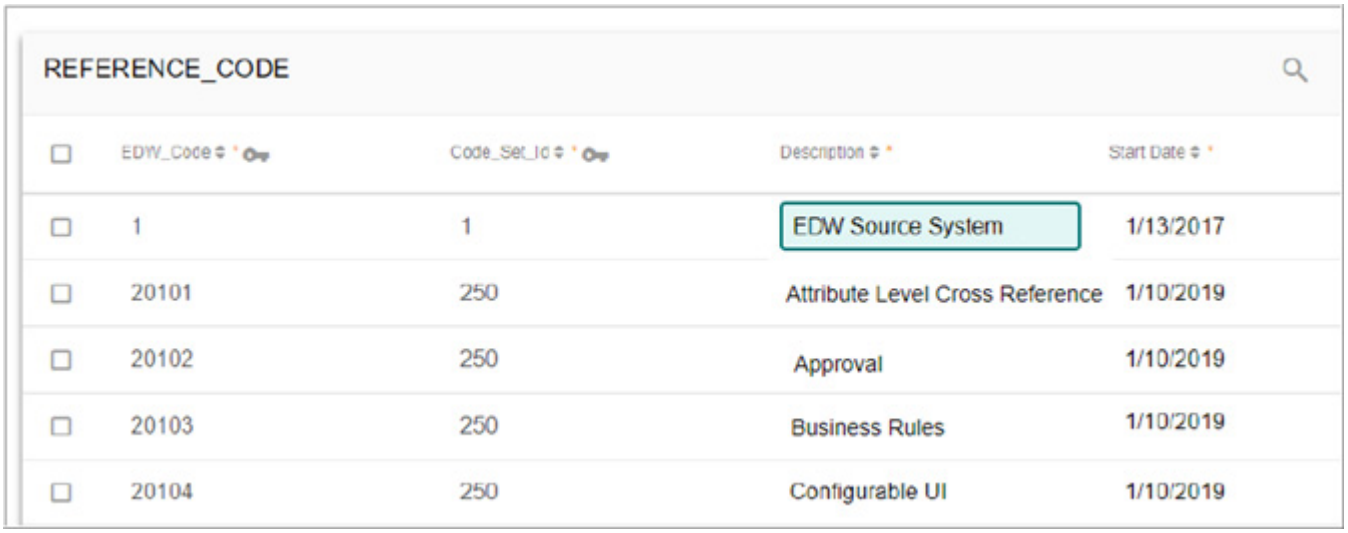

- **Inline Edit on Data Table Fields**

  Using the Inline editing as in Figure 81, you can update and save the record details without navigating from the current page. OOB the inline editing is not enabled on the on data table UI fields. To enable inline editing, set inlineEdit="true" in <ui:table>.

  ```
  <ui:table inlineEdit="true">
  ```

  Inline editing can also be enabled on the MDM UI from UI's like Generate Configurable UI, Model Builder UI. For details, refer to MDM 4.6 Server Guide.pdf.

Figure 81: Inline Editing




- **Breadcrumbs, print and refresh disabled by default.**

  To display the print, refresh, use 'showHeader' attribute in <ui:page-group> as below.

  ```
  <ui:page-group showHeader="true">
  ```

- **Inline Menu option**

  To Enable data table inline actions as in Figure 82.

  - For single row selection data table

    Mention inlineActions="true"(Default is true) to <ui:table>

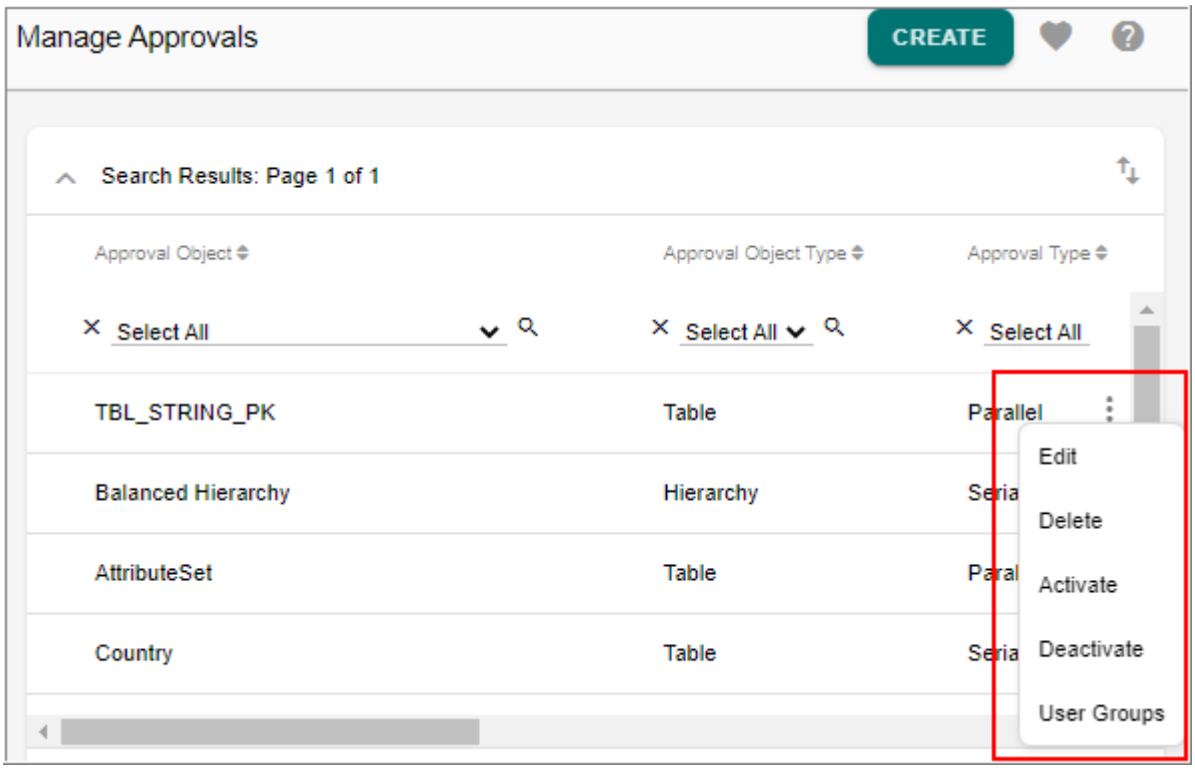
    Mention inlineAction="true"(Default is false) or enabledBySelection="true"(Default is false) to <ui:button>

  - For multi row selection data table

    Mention inlineActions="true"(Default is false) to <ui:table>

    Mention inlineAction="true"(Default is false) to <ui:button>

Figure 82: Inline Menu Option

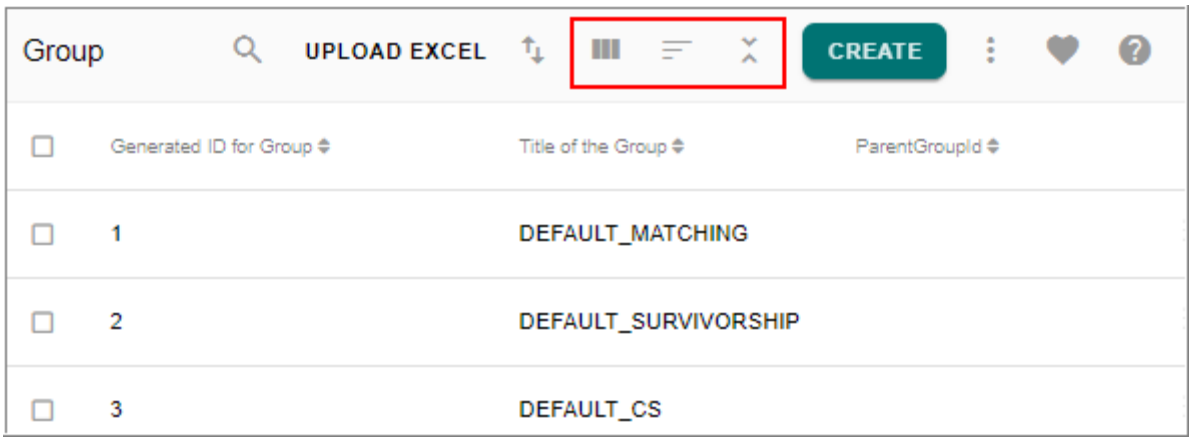



- **Disable Header Icons**

  Header icons like customize, export etc. can be disabled for data table using showHeader="false" in <ui:table> as in Figure 83.

  ```
  <ui:table showHeader="false">
  ```

Figure 83: Header Icons (Customize, Multi-Column Sort, Export)




Add below code under <ui:header> to enable required icons/buttons

- For customize icon

  ```
  <ui:image-link-field name="customizeTable" src="pgl-icon mat-icon
  view_column" title="Customize Table"
  onClick="javascript:ui_configureTableV2('searchResults');"
  id="customizeTable"/>
  ```

where 'searchResults' is 'id' of table

- For export button

```
<ui:button tableButton="true" id="Export" name="Export"
displayText="Export"
onClick="javascript:table_export('searchResults','false','ui');"/>
```

where 'searchResults' is 'id' of table

Also add below code under <ui:buttons> of <ui:table>

```
<ui:button headerButton="true" id="Export" name="Export"
displayText="Export" enabledBySelections="true"/>
```

- For multisort icon

```
<ui:image-link-field name="toggleMultiSort" src="pgl-icon mat-icon
sort" title="Enable/Disable Multi Column Sort"
onClick="javascript:toggleMultiSortUIFlag('searchResultsMultiSortI
con');" id="toggleMultiSort"/>
```

where 'searchResultsMultiSortIcon' is 'id' of below multi sort icon

```
<ui:image-link-field style="display:none"
name="searchResultsMultiSortIcon" src="pgl-icon mat-icon
unfold_less" title="Multi Sort"
onClick="javascript:showMultiSortOrderMenu();"
id="searchResultsMultiSortIcon"/>
```

- **Enable/Disable Vertical Scroll Bar**

  By default, vertical scroll bar is enabled at table level. Scroll bar at table level can be disabled by setting the attribute verticalScroll="false" in <ui:table> as below. Once this attribute is set to false, the page level scroll bar will be enabled by default.

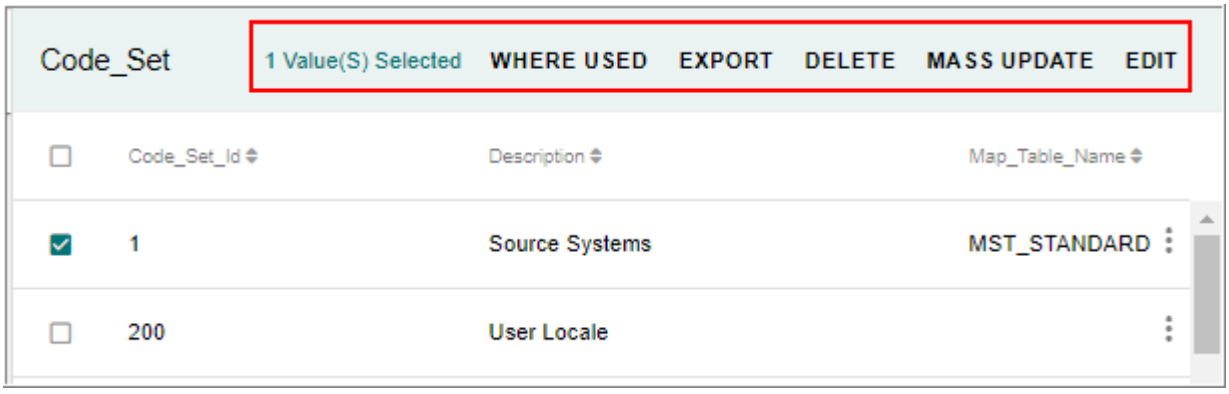
```
<ui:table verticalScroll="false">
```

  Specify 'width' to <ui:cell> to align scroll bars correctly. (<ui:cell width="100%">)

- **Button on demand**

  On selecting a record, the relevant buttons will be displayed as in Figure 84. This capability can be set using the 'enabledBySelections' attribute in <ui:button>.

Figure 84: Sample UI for Card View

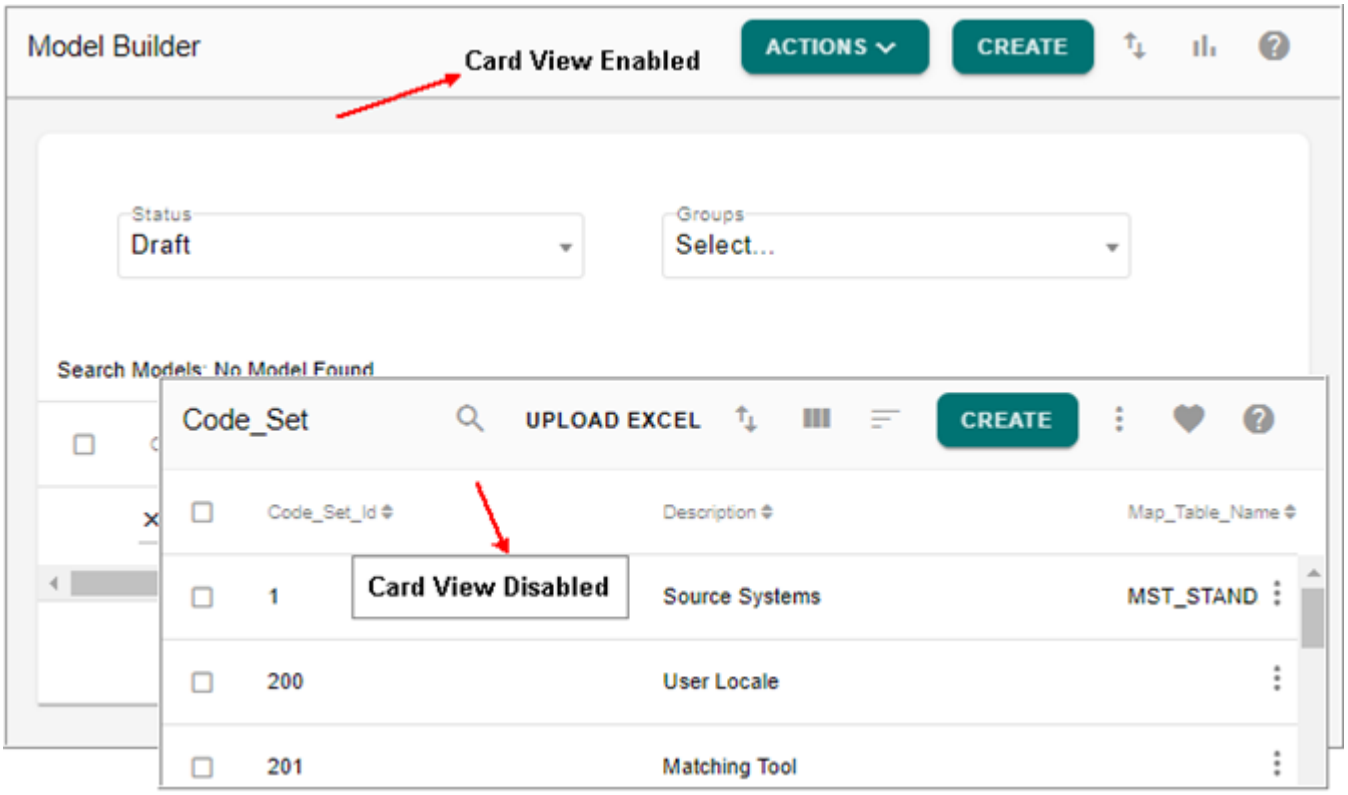



- **Disable Card View**

  Card view can be disabled on the top container as in Figure 85 using cardView="false".

```
<ui:container cardView="false">
```

Figure 85: Sample UI for Card View




## Forms Improvements

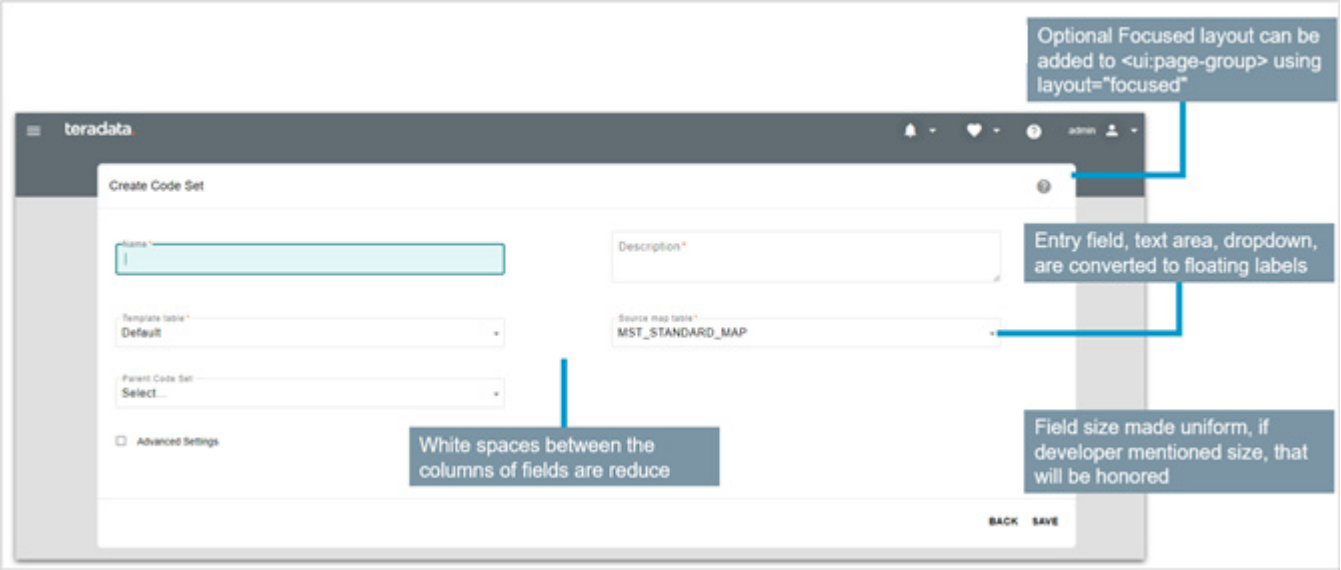
Improved forms layout by reducing lot of white spaces, maintained uniform field size, retained only the relevant fields on the main form area and moved optional text fields under Advanced Settings option as in Figure 86.

- **Focused layout** can be added as in Figure 86 to <ui:page-group> using layout="focused" .

  The layout attribute value can be added for <ui:page-group> only.

  ```
  <ui:page-group layout="focused">
  ```

Figure 86: MDM Sample Form UI



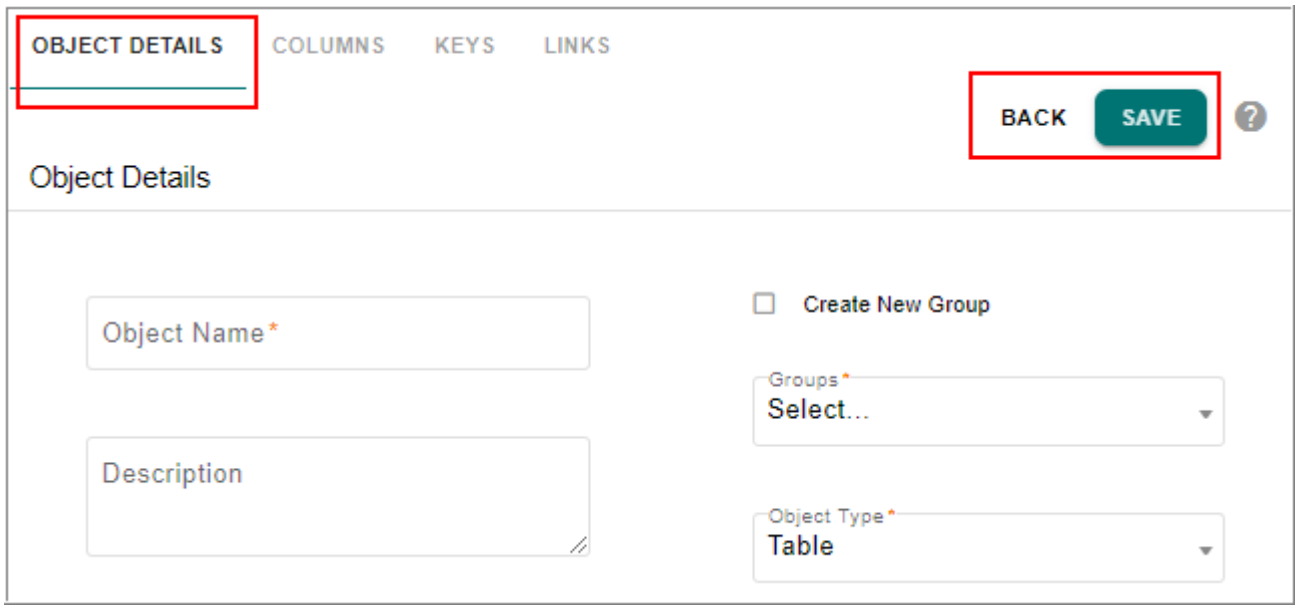

- Labels for entry field, text area, dropdown are converted to floating labels. Longer floating labels are truncated as in Figure 86. These styles will be applied by default and no changes required from the developer.
- New 'style' attribute added to <ui:field-group>. (<ui:field-group style="padding:0px">)

  The above style attribute can be used for border padding, margin, color etc.
- **Tabs**: To show/hide button menu respective to current tab as in Figure 87.

  Under each tab, the buttons to be displayed can be set using the ‘displayCondition' attribute in <ui:button> component as in below code sample

  ```
  <ui:button id="save" name="save" displayText="SC_SAVE"
  tableButton="true" emphasized="true" displayCondition="{$root/tab/
  @Value='OBJECT_DETAILS'}" onClick="javascript:onSave();"/>
  ```

Figure 87: Sample UI for Button Display under each Tab




## General Guidelines

The below section describes few good coding practices:

- Complete PGL code should be within one <ui:container>.

  As a good coding practice and framework guideline, all the PGL code must be written within one <ui:container>

  ```
  <ui:container id="container1" name="container1">
      <ui:step id="Outer" name="Outer" displayText="Saved Filters">
          -----
          -----
      </ui:step>
    </ui:container>
  ```

- Do not use nested <ui:field-group> elements unless required.

  It is recommended not to write nested code in <ui:field-group> that is nesting <ui:field-group> within another <ui:field-group> and reduce the usage of <ui:grid> as a good coding practice.

- To remove extra white spacing between the fields, use the space attribute in <ui:field> as <ui:field-space> .

- Success/Error messages: For any extra space between icon and message then add 'width:auto' to <ui:field-group>

  ```
  <ui:field-group style=”width:auto”>
  ```

# APPENDIX A Using the PGL Example Project

### What's In This Appendix

This appendix walks you through the usage of the PGL example project. It includes the following topics.

### Related Topics


- Introduction
- Location
- Workflows
- Running Workflows
- MDM Pop-ups


## Introduction

If you have successfully installed MDM, you should be able to find the example project as one of the sample projects with the install.

This appendix includes the following information:

- The location of the project, starting up the project and configuration. For details see "Location".
- The workflows in the project and aspects of PGL covered in the workflows. For details see "Workflows".
- Running the workflows, creating your own workflows using the example project, and the UI reference manual which is a part of the install. For details see "Running Workflows".

## Location

The PGLExamples folder includes the configuration files (myServer.xml and myService.xml in the `cfg` directory), rule files (`rules` directory), workflow and PGL files (`workflow` directory).

To open the project:

1. Launch the studio.
2. Open the PGLExamples.owb present in the following location:

   `<STUDIO_HOME>\services\studio\samples\PGLExamples\`

**Note:** You need not specify the database configurations. The project does not require a database as it works on data specified in the workflows.

# Workflows

The PGL Example project consists of the workflows covering the following sections:

- Basic Controls
- Advanced Controls
- Samples

The workflows have been categorized to provide easy understanding and easy navigation. Refer to the UINavigationRule.xml file in the project to understand the structure of the left navigation pad items.

The left navigation has been organized using the structure given below.

**Note:** The workflow and PGL files are available in the workflow folder in the `PGLExamples` folder.

## Basic Controls

The Basic Controls workflow demonstrates the usage of most of the basic controls in PGL. The following is the list of PGLs in the Basic Controls workflow:

Table 96: Basic Controls

| Left Navigation Item | PGL Name | Controls Used |
|---|---|---|
| Basic | 1_Basic | Display Field, Entry Field, Link Field, Text Box, Check Box, Radio Button, Drop Down, List Box |
| Container | 2_Container | Containers |
| Image | 3_Image | Image, Image Link |
| Chart | 4_Chart | Chart, Chart Field |
| Tab | 5_Tab | Step |
| Wizard | 6_Wizard | Step |
| Field Groups | 7_Field_Groups | Field Groups |
| Table/Simple | 8a_Table_Simple | Table with Paging, Table without Paging, Invert Table, Export to Excel |
| Table/Auto Fields | 8b_Table_Auto_Fields | Table with Auto Fields |
| Table/Tree Cell | 8c_Table_Tree_Cell | Table with Tree Cell |

For details on all the available basic controls see “Chapter 4 Basic Controls” on page 28.

## Advanced Controls

The Advanced Controls workflow demonstrates the usage of most of the advanced controls in PGL. The following is the list of PGLs in the Advanced Controls workflow:

Table 97: Advanced Controls

| Left Navigation Item | PGL Name | Controls Used |
|---|---|---|
| Conditional | 1_Conditional | Uses Conditional Fields in Table with ui:when and ui:otherwise |
| Tree Grid | 2_Tree_Grid | Vertical Tree Browser with Grid and Cell |
| Pivot | 3_Pivot | Pivot Table |
| Excel Pivot | 4_Excel_Pivot | Excel Pivot Table |
| Dynamic | 5_Dynamic | Dynamic PGL |
| Horizontal Tree | 6_Horizontal_Tree | Horizontal Tree Browser |

For details on all the available basic controls see “Chapter 5 Advanced Controls” on page 150.

## Samples

The Samples workflow demonstrates the mechanics of using of a combination of PGL components. The following is the list of PGLs in the Samples workflow:

Table 98: Samples

| Left Navigation Item | PGL Name | Controls Used |
|---|---|---|
| Search | 1_Search | Search and display results in a table |
| Preview | 2_Preview | Browse, Attach and Preview images and Files |
| Advanced Tables | 3_Tables_Advanced | To use column filters |
| Mass Entry | 4_Mass_Entry | To use multi select in a table and collate results |
| Advanced WIzard | 5_Wizard_Advanced | To move between steps of a wizard |
| Popups | 6a_Popup_Main and 6b_Popup_Receive | To pass data to and from a popup |

# Running Workflows

Perform the following steps to run the project:

To run the PGLExamples project:

1 Start Studio Locator.

2 Launch Studio and open the PGLExamples.owb project file available at the following location.

```
<INSTALL_HOME>\studio\Samples\PGLExamples\
```

3 Start Local Server within Studio.

4 Start Weblogic Server.

Wait till you see the following message on weblogic console:

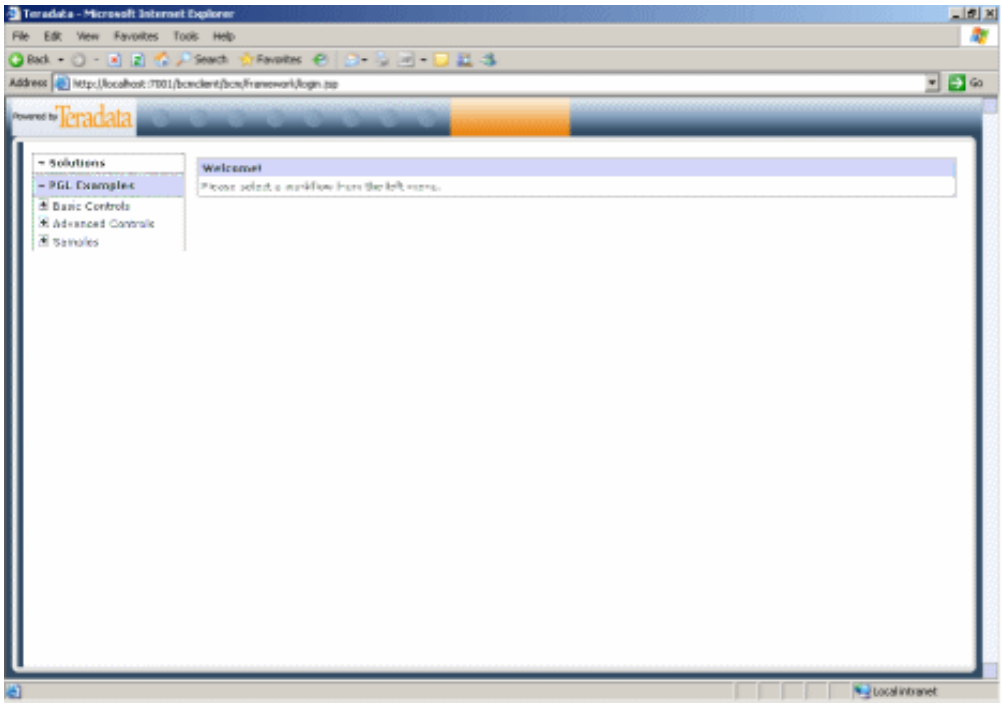
```
<DATE TIME> <Notice> <WebLogicServer> <BEA-000355> <Thread
"ListenThread.Default" listening on port 7001, ip address *.*>
```

5 Open the URL, http://localhost:7001/base/core/login.jsp.

6 Login with User ID/Password as admin/admin.

The left pane will show the navigation links for the project

Figure 88: PGL Examples main

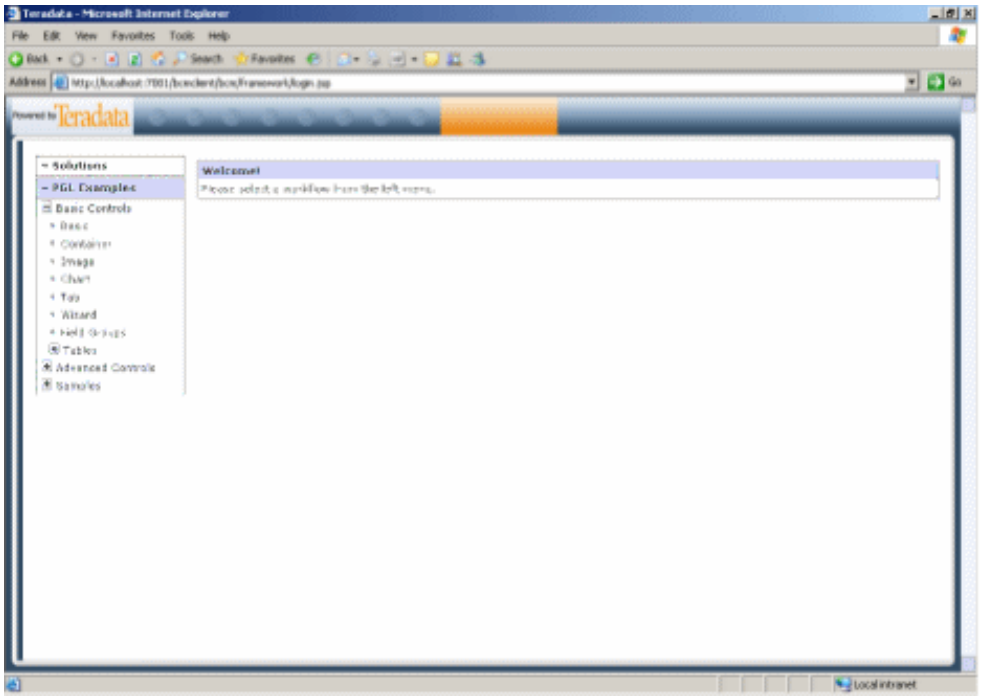



Navigate through the links in the left navigation pane to run the workflows. Follow the instructions on the screens when you are running the workflows.

To understand how each component works, use the Debugger in studio. Set break points in the workflow and examine the values taken by variables as you run through the workflows.

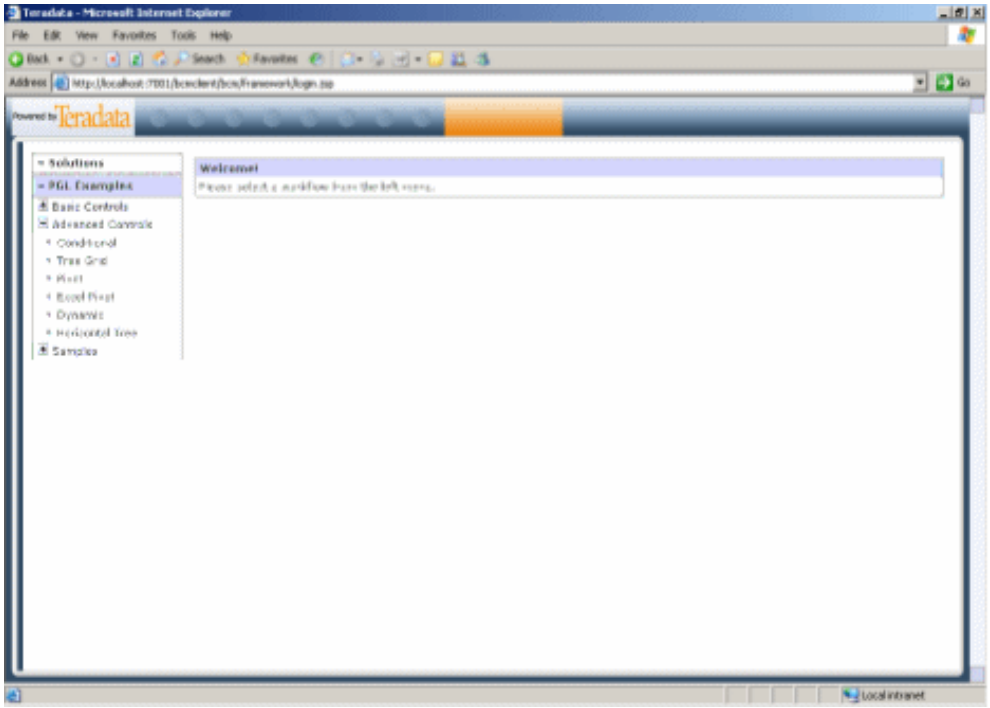
The following figure shows the list of all the Basic Controls available in the workflow.

Figure 89: Basic Controls



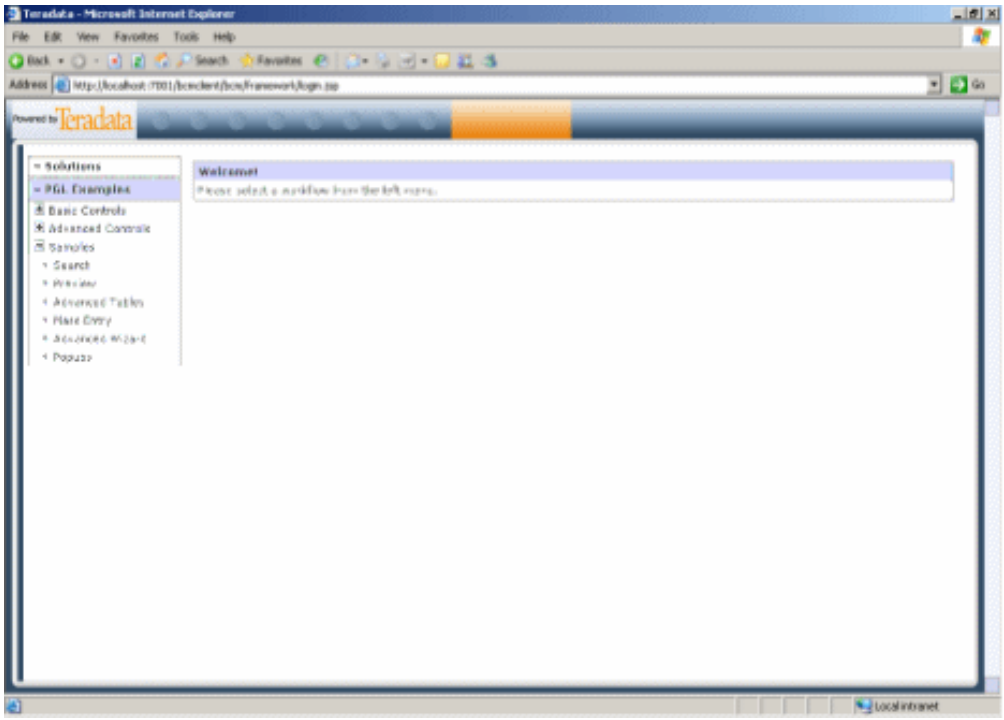

The following figure shows the list of all the Advanced Controls available in the workflow.

Figure 90: Advanced Controls




The following figure shows the list of all the Samples available in the workflow.

Figure 91: Samples




## Example

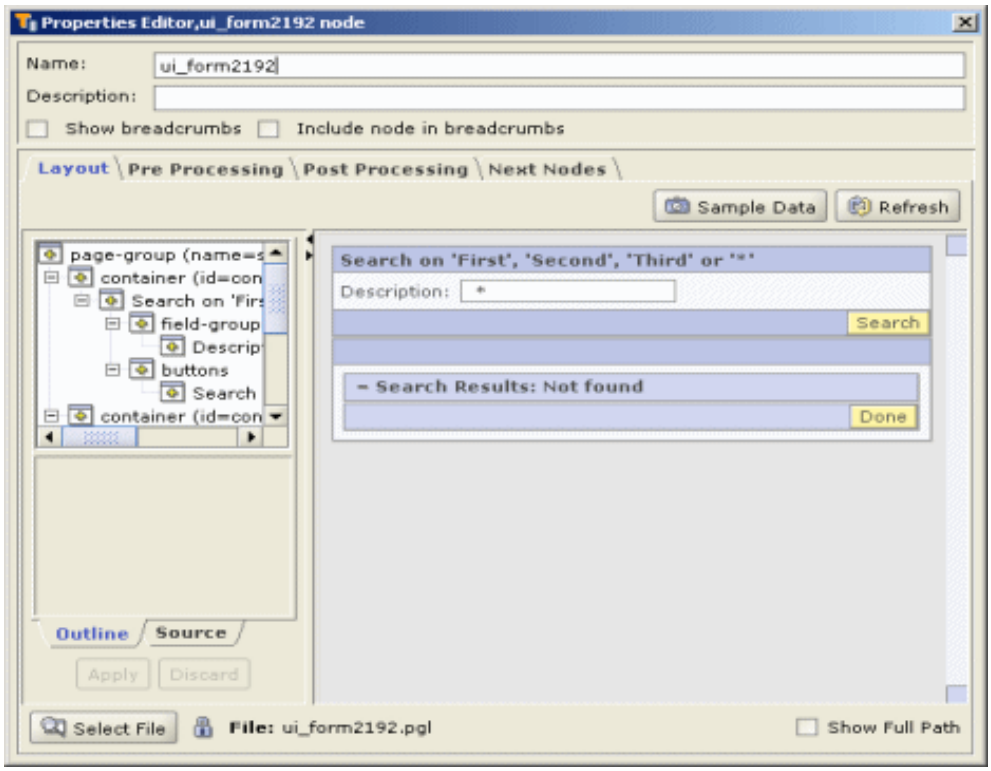
The following steps walk through one of the workflows to understand the usage of the project.

To understand the usage of the project:

1. Open Samples.xml workflow in the studio. This workflow is to demonstrate usage of the search and display feature.
2. Double click on the Search UI node.

   The Properties editor with the 1_Serach PGL is displayed.

Figure 92: Properties Editor




The 1_Search PGL has one entry field to specify the search parameters and a table to display the search results. The entry field value is * by default.

3 In the Samples.xml workflow, there is a task node called `preXmlSearch` connected to the UI node. The `preXmlSearch` node contains the data in a pipeline variable (`$preXml1`), that is to be displayed in the search results table, depending on your search condition.

**Note:** If the data exists in the database, the task node in the workflow will contain a `GET_DOCUMENT`, `EXEC_SQL_QUERY` etc.

4 Depending on your search conditions First, Second, Third, or *, the Post – action will set appropriate values of `$populateSearchResults` and `$searchfilter`.

For example,

```
<IF_TEST Test="$userAction='SEARCH' and $outXml/Description/
@Value='*'">
   <THEN>
      <SET Var="populateSearchResults" FromValue="yes"/>
      <SET Var="searchfilter" FromValue="xmlAll"/>
   </THEN>
</IF_TEST>
```

5 Depending on the value of `$searchfilter` which is a pipeline variable, `$inXml` value in pre-action will be set correctly.

For example,

```
<IF_TEST Test="($populateSearchResults/@Value='yes') and
($searchfilter='xmlAll')">
<THEN>
<TO_DOCVAR AssignToVar="inXml">
  <SAMPLE_XML>
    <TO_XML SelectList="$preXml1/SAMPLE_XML_All/*"/>
  </SAMPLE_XML>
</TO_DOCVAR>
</THEN>
</IF_TEST>
```

6 Switch to the source mode of the workflow

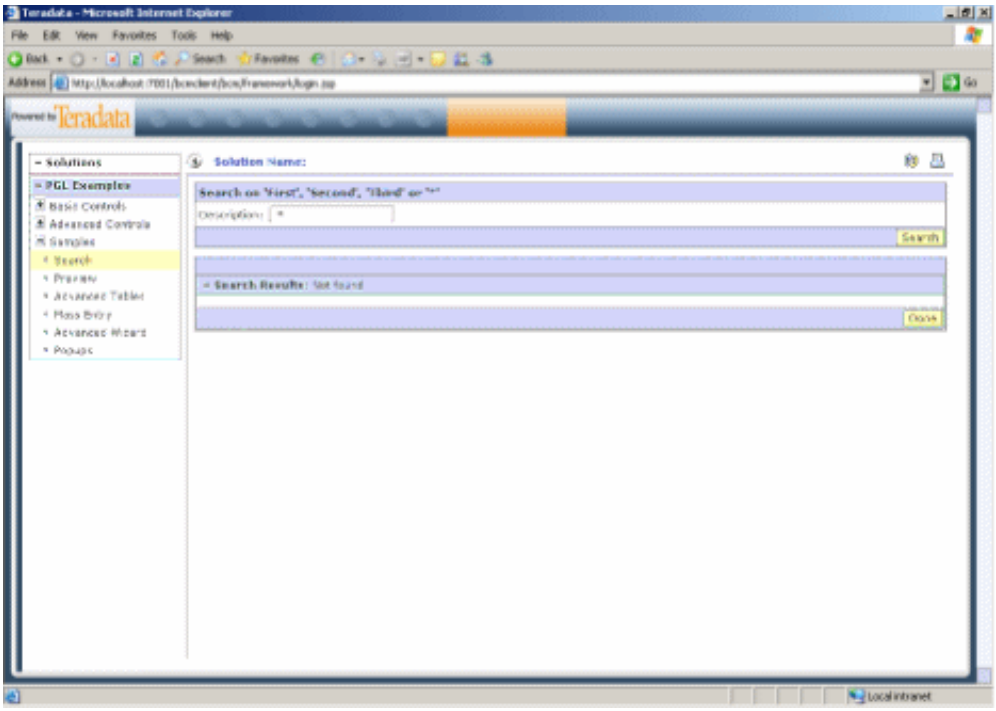
7 Set break points in pre and post processing steps of the workflow

8 Login to the UI using the URL, http://localhost:7001/base/core/login.jsp.

9 Navigate to **Samples** > **Search**.

10 Click **Search**. Control shifts to pre-action. Run through the workflow. The following screen will be seen.

Figure 93: Search

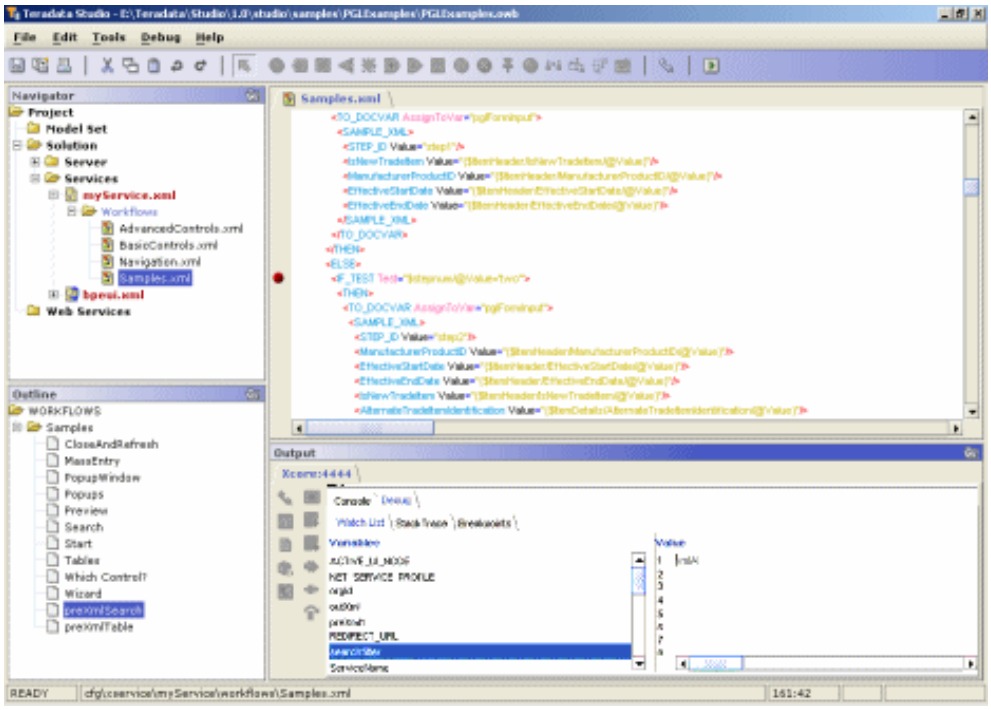



The search condition is set to '*' by default.

11 Click **Search**. Control shifts to the break point in post action in studio.

12 Step into and check the values of `searchfilter`, `populateSearchResults` and `outXml`.

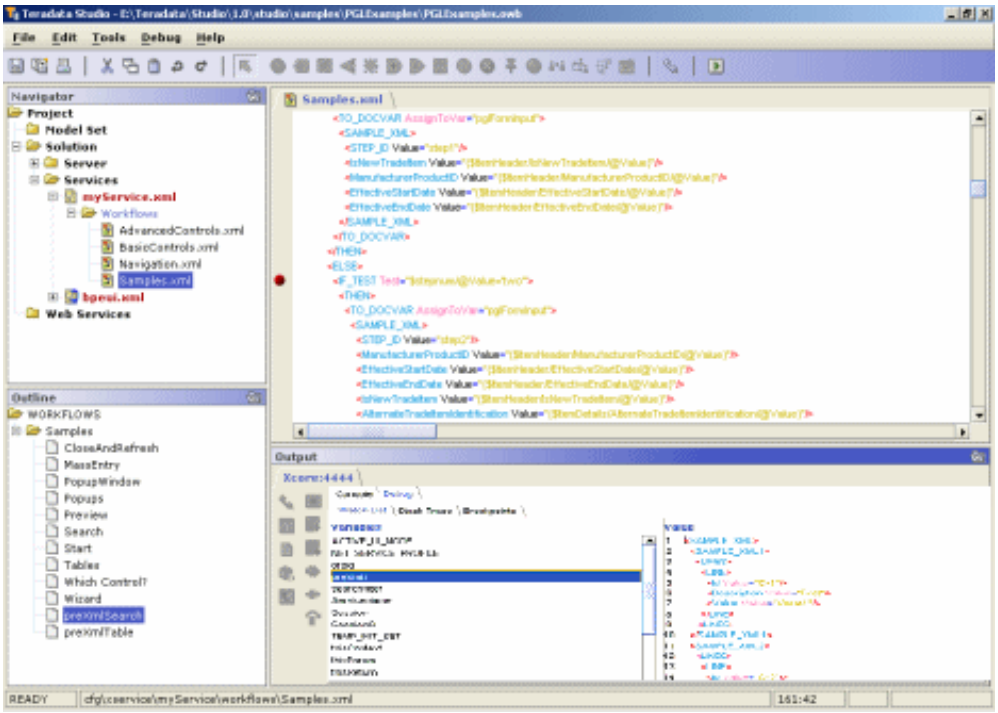
Figure 94: Search



13 Run to the next break point. Control shifts to pre-action to the `IF_TEST` condition and inXml is set accordingly.

```
<IF_TEST Test="($populateSearchResults/@Value='yes') and
($searchfilter='xmlAll')">
```

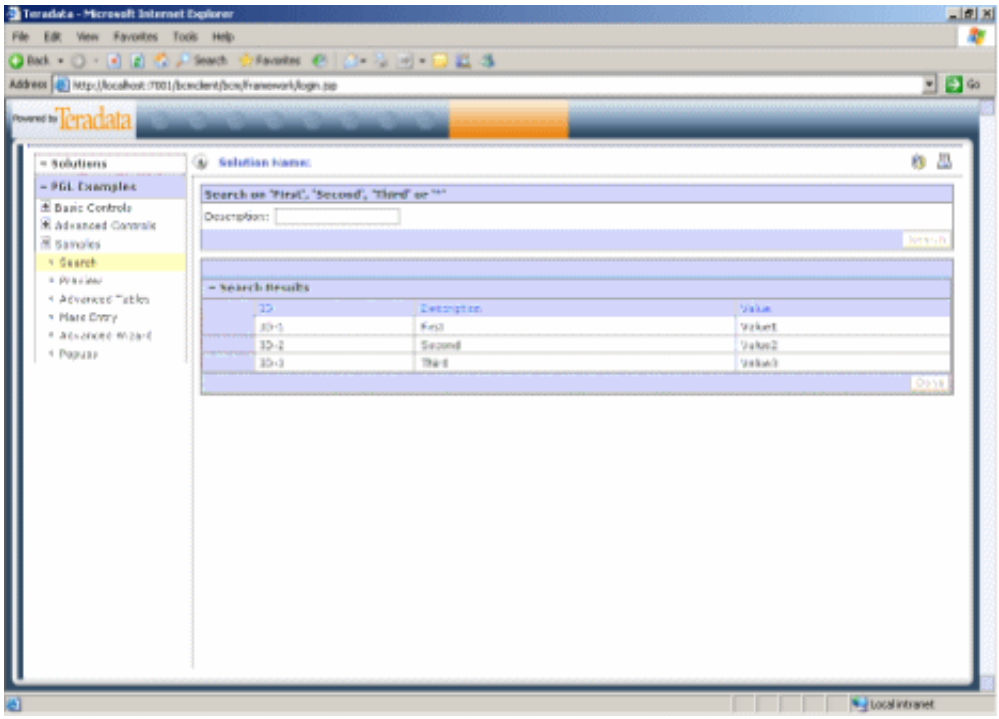
Figure 95: Search



14 Run through the workflow and go back to the UI.

The search results table is populated with the search results.

Figure 96: Search Complete




# MDM Pop-ups

Pop-ups were supported in MDM 3.3.1 or earlier versions using two main methods namely core_alert and core_confirm. The above methods needs to be updated as core_alert_jquery and core_confirm_jquery for proper browser compatibility as in the following sections.

**Core_alert**:

The core_alert method needs to be changed as core_alert_jquery and core_alert_jquery method does not require any call back method name.

For example:

Existing code:

```
function onDelete()
{
core_alert('Please select at least one Role');
}
```

The above block of code needs to be replaced as below:

```
function onDelete()
```

```
{
core_alert_jquery('Please select at least one Role');
}
```

**Core_confirm**:

The core_confirm method needs to be changed as core_confirm_jquery and core_confirm_jquery method needs an extra parameter for call back method name. User nominated option will be passed as parameter to call back method.

For example:

Existing code:

```
function onDelete()
{
if( core_confirm('Are you sure you want to delete the selected
Approval?') == 'yes' )
{
document.form.BUTTON_ID.value = 'DELETE';
document.form.action = 'view.x2ps';
document.form.submit();
grayOut(true);
}
}
```

The above block of code needs to be replaced as below:

```
function onDelete()
{
core_confirm_jquery('Are you sure you want to delete the selected
Approval','onDeleteCallback');
}
function onDeleteCallback(response)
{
if( response == 'yes' )
{
document.form.BUTTON_ID.value = 'DELETE';
document.form.action = 'view.x2ps';
document.form.submit();
grayOut(true);
    }
}
```

# APPENDIX B Hierarchy Viewer-Custom Workflow

### What's In This Appendix

This appendix explains the different approaches to loading a hierarchy in custom workflows.

### Related Topics

- Implementation Options

# Implementation Options

Hierarchy is loaded using the ViewHierarchyJSTree workflow. This workflow can be called via URL from any other workflow to load a particular hierarchy. It is also important to pass the information related to the hierarchy being loaded as parameters along with the URL. A sample URL is shown below:-

```
var newURL =  <context_path> +   "/
start.x2ps?START_WORKFLOW=ViewHierarchyJSTree&SERVICE_NAME=UI_WORKFL
OW&SOLUTION_NAME=MDM&SelectedRowId=HIERARCHY_ID=<HID>%23NAME=<HName>
%23HIERARCHY_TYPE=<HType>&editable=<isEditable>"
```

In the above URL:

- <context-path> should be replaced with the variable which contains the context-path.
- Parameter 'SelectedRowId' contains the information regarding the hierarchy being loaded. Here <HID> should be replaced with Hierarchy ID, <HName> with the name of the hierarchy,<HType> with the type of the hierarchy and <isEditable> with 'yes' or 'no' depending on whether the hierarchy is editable or not.
- There are four optional parameters that can be provided along with the URL:
  - dynamicFilter : One can provide the dynamic filter if any, as value to this parameter and pass it along with the URL. This is applicable only if the Relational Objects involved in the hierarchy expects a dynamic filter value.
  - AsOfDate: This helps in providing As of date when the hierarchy being loaded is a temporal hierarchy.
  - HIERARCHY_VERSION: If the hierarchy being loaded is a versioned hierarchy then the version id should be provided as value to this parameter.
  - HideBreadcrumb: If the breadcrumbs in the hierarchy viewer workflow should be hidden set the value of this parameter to 'true

  Below are some sample URLs with the above mentioned optional parameters:

- /mdm/start.x2ps?START_WORKFLOW=ViewHierarchyJSTree&SERVICE_NAME=UI_WORKFLOW&SOLUTION_NAME=MDM&SelectedRowId=HIERARCHY_ID=2%23HIERACHY_NAME=test%23HIERARCHY_TYPE=R&editable=no&HideBreadcrumb=true&HIERARCHY_VERSION=202%23v6

  **Note:** Value of parameter 'HIERARCHY_VERSION' is 202%23v6. '#' is encoded as '%23' in URL path. '202' is the Hierarchy Version ID and 'v6' is the version name.

- /mdm/start.x2ps?START_WORKFLOW=ViewHierarchyJSTree&SERVICE_NAME=UI_WORKFLOW&SOLUTION_NAME=MDM&SelectedRowId=HIERARCHY_ID=10%23HIERACHY_NAME=tempHier%23HIERARCHY_TYPE=R&editable=no&dynamicFilter=1119||1126%232&AsOfDate=06/16/2015%2019:10:00

  **Note:** Value of parameter 'dynamicFilter' is '1119||1126%232'. '#' is encoded as '%23' in URL path. '1119' and '1126' are the Relational Object IDs involved in the hierarchy and '2' dynamic filter value for the Relational Object '1126'. Also value of parameter 'AsOfDate' is '06/16/2015%2019:10:00' where '%20' implies space character, '06/16/2015' is the date and '19:10:00' is the time.

There are two ways of implementing the hierarchy viewer in custom workflows:

**a** The URL mentioned above can be loaded in an iframe within any desired page. For example, create a div tag with id as 'div1' and load the iframe within it as below:

$("#div1").html(" <iframe id='frame1' width='99%' height='99%' frameborder='1' src='"+<newURL>+"'/>");

<newURL> should be replaced with the appropriate URL to be loaded as explained earlier.

A limitation with this approach is that since its loading the workflow within another workflow, once the hierarchy is loaded, the workflow in which it was loaded cannot be used further as its workflow state would be lost. For example, after the hierarchy workflow is loaded in a parent workflow, the parent workflow cannot be refreshed or be used for any other processes. But the hierarchy can be used further without any other issues.

**b** Another approach is loading the hierarchy workflow in a different page. That is, from a custom workflow when a button is pressed or when any other event is triggered, the page can be redirected to the URL mentioned (hierarchy workflow). A sample code on how this can be achieved is shown below:

```
 function onEventTrigger() // The function to be called when the
page is to be redirected
  {
     document.form.action = <newUrl>;
     document.form.submit();
  }
```

<newURL> should be replaced with the appropriate URL to be loaded as explained earlier.

In this approach once the page is redirected, the control will be transferred from the custom workflow to hierarchy viewer workflow and there is no way to come back to the custom workflow unless that provision is added in the hierarchy viewer workflow.

# APPENDIX C Timer Service

### What's In This Appendix

This appendix provides information on using the Timer Service.

Topics include:


- Introduction
- Timer Setup and Callback
- Configuration of Timer Poll Interval
- Timer Commands
- Timer Functions
- Timer Examples
- Guaranteed Messaging


# Introduction

Timer service allows you to perform specified actions at a specified time. It also allows you to specify a duration at which the timer should repeat these actions. It provides you with a collection of functions and tags to perform the timer related activities.

Timer Service operates in the following two phases:

1. The setup of a timer in the system with a specified callback time. This activity is handled by the xcore server and it takes care of adding timer records to the database.
2. The Timer Sink runs as a service on the xcore server and takes care of performing the specified callback actions on the services. It also takes care of repeating the call after the specified duration.

## Deployment

In a typical deployment there will be multiple Xcore instances. Each of these instances will be homogeneously deployed with multiple Xservices. Each of the Xcore instances will have a timer source component and a timer sink component. So there will be multiple timer services running across different Xcore instances. When a service in one xcore instance starts a timer job, the xcore server will persist the job as a timer record in the database. Whenever this job is due to be run, any of the timer sink services across xcore servers can pick up this job and act on it. So, timer callback actions can be executed in an xcore instance that is not the one that initiated it.

# Timer Setup and Callback

Services can define a timer by specifying the timer name, call back time, call back actions and optionally a document or a duration after which it should repeat the call. This gets persisted as a timer record in the database with a calculated expiration date.

The timer sink service polls the database periodically to select timer records that have expired. The interval at which the service polls the database is configurable. For every selected expired timer record, the timer service marks a claim date. This prevents another timer sink service of another xcore server, to act upon the expired records. For each of these timer records, callback actions are executed.

- For a *simple timer callback request*, the service attempts callback once and then removes the timer record from the database.
- For a *durational timer*, the expiration date is incremented using the duration that is specified and the timer record in the database updated.
- For a *guaranteed timer request*, the timer service will retry for the specified number of times at specified time intervals. It will then execute the callback actions specified upon success or failure as the case may be.

To define timers, delete timers or to get an existing timer details, timer commands are used.

**Note:** NUSI have been removed from TIMER table due to deadlock issue. If you want to recreate indexes, execute the below SQL using Teradata query tool.

```
CREATE INDEX TIMER_I0 ( EXPIRATION_DATE ,CLAIM_DATE ) ON TIMER ;
CREATE INDEX TIMER_I1 ( EXPIRATION_DATE ) ON TIMER;
CREATE INDEX TIMER_I2 ( CLAIM_DATE ) ON TIMER;
CREATE INDEX TIMER_I3 ( TIMER_ID )ON TIMER ;
```

# Configuration of Timer Poll Interval

There are two ways to configure a timer poll interval

1. In timersink.xml, the property file for the timer sink service, you can change the value specified and then bring up the xcore server.

   ```
   <SERVICE-PARAMS>
      <PARAM Value="40" Name="TIMER_SLEEP_TIME" Comment="Specify
      sleep time in seconds"/>
   </SERVICE-PARAMS>
   ```

2. Through command line utility on the xcore server console at runtime.

   a. `Command xcore>GetTimerFreq`

      This will return the timer poll that is currently set to the specified interval in seconds.

      For example, `Xcore>40`

b `Command xcore>SetTimerFreq 500`

This will change the timer poll that is currently set, to the specified interval in seconds.

# Timer Commands

## START_TIMER

Description: It adds a timer record to the database identified by the given name.

The following are the pre-defined variables that can be referenced within the timer call back rule:

- thisDoc: The name of the document you specify in the `START_TIMER` through the `FOR_DOCUMENT` tag.
- identifiedByDoc: This is the identifier.
- documentcallBackDateVar: This is the expiration date time for which the timer is being invoked (`CALLBACK_DATE`).
- callBackDurationVar: This is the call back duration parameter which is set in the `CALLBACK_DURATION`.
- callBackDateVar: This is a special variable that stores the call back date time. It can be used in call back actions.

For example,

```
<CALL_BACK_ACTIONS>
  <PRINT Value=”Call back date time is : “  Nl=”yes” />
  <PRINT Var=”callBackDateVar” />
</CALL_BACK_ACTIONS>
```

The call back actions are evaluated only when the timer document expires. All the variables, x-path expressions are evaluated when the call back actions are evaluated. So it is important to realize that you cannot use any of the variables in the parent rule invoking the `START_TIMER` tag. You are limited to the pre-defined variables described above and any other variables defined with in the scope of `CALLBACK_ACTIONS` tag.

Syntax:

```
<START_TIMER>
  <!--The timer record will be identified by this name. This is
  mandatory but does not have to be unique. -->
  <IDENTIFIED_BY>
     <NAME Value="{$Timer Name}"/>
  </IDENTIFIED_BY>
  <!--the document for call back actions-->
  <FOR_DOCUMENT>
     <DOC_NAME>
```

```
        <DOC_DATA/>
      </DOC_NAME>
   </FOR_DOCUMENT>
   <!-- Call back repeat duration. Duration function can be used to
   get number of seconds. -->
   <CALLBACK_DURATION Select="{$number of secs after which to call
   back}"/>
   <!--The datetime when the actions should be first performed. Date
   function can be used to get the date time.-->
   <CALLBACK_DATE Select="{$datetime}"/>
   <!--The actions to be performed when the timer goes off.-->
   <CALLBACK_ACTIONS>
      <PRINTLN Value="thisDoc Variable = {$thisDoc}"/>
      <PRINTLN Value="Doc data = {$thisDoc/DOC_DATA}"/>
      <PRINTLN Value="Identifier name = {$identifiedDoc/NAME/
      @Value}"/>
      <PRINTLN Value="Current Expiration Date = {$callBackDateVar}"/>
      <PRINTLN Value="Timer Duration = {$callBackDurationVar}"/>
      <SET  Var="myVar" FromValue=" {$callBackDurationVar}"/>
   </CALLBACK_ACTIONS>
</START_TIMER>
```

Attributes:

None

## STOP_TIMER

Description: Deletes the timer record identified by the given name or Id.

Syntax:

```
<STOP_TIMER>
  <IDENTIFIED_BY>
     <NAME Value="{$timerName}"/>
  </IDENTIFIED_BY>
</STOP_TIMER>
```

OR

```
<STOP_TIMER>
  <ID Value="{$timerRecordId}"/>
</STOP_TIMER>
```

# Timer Functions

The date function can be used to specify the call back datetime. The function returns the current date with time stamp if used without any parameters. It can be used to build any date using different parameters. For details on timer functions see "Time Functions" on page 402.

# Timer Examples

The following are examples on timers.

*Example 1:*

Description: The following example defines a timer with name `SIMPLE_TIMER`. It asks to call back after six hours. When it calls back it prints the date time and calls timerCallBackHandler request.

```
<DEFINE_METHOD Name="startSimpleTimer">
<RULE>
<ACTION>
<START_TIMER>
  <IDENTIFIED_BY>
    <NAME Value="SIMPLE_TIMER"/>
  </IDENTIFIED_BY>
  <CALLBACK_DATE Value="{incrDate(date( ),duration
  (0, 6, 0,0))}"/>
  <CALLBACK_ACTIONS>
    <PRINTLN Value="Call back date time is: "/>
    <PRINTLN Value="{callBackDateVar}"/>
    <REQUEST Name="timerCallBackHandler "/>
  </CALLBACK_ACTIONS>
</START_TIMER>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

*Example 2:*

Description: The following example defines a timer. It passes the parameters sent to the method, to the actions. It calls back after one hour and repeats call back after one hour. When

it calls back it prints the parameters for the call back actions and executes myRequest method with the parameters.

```
<DEFINE_METHOD Name="startForDocumentTimer">
<RULE>
<ACTION>
<START_TIMER>
  <IDENTIFIED_BY>
    <NAME Value="FOR_DOCUMENT_TIMER"/>
  </IDENTIFIED_BY>
  <FOR_DOCUMENT>
    <TO_XML DocVar="thisParam"/>
  </FOR_DOCUMENT>
  <CALLBACK_DURATION Value="{duration(0,1,0,0)}"/>
  <CALLBACK_DATE Value="{incrDate(date(),
  duration(0, 1, 0, 0)))}"/>
  <CALLBACK_ACTIONS>
    <PRINT Value="Request parameters for the
    actions are:"/>
    <PRINT Var="thisDoc"/>
    <REQUEST Name="myRequest">
      <TO_XML DocVar="thisDoc"/>
    </REQUEST>
  </CALLBACK_ACTIONS>
</START_TIMER>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

*Example 3:*

Description: The following example deletes any timer instances with name `INVOICE_TIMER` and starts a new timer with this name. It asks to call back at 11:59:59 PM the previous day and to repeat calling every day at this time. When the timer calls back it will execute request startInvoiceRun with the call back date/time as parameter.

```
<DEFINE_INIT>
<RULE>
```

```
<ACTION>
<STOP_TIMER>
  <IDENTIFIED_BY>
    <NAME Value="INVOICE _TIMER"/>
  </IDENTIFIED_BY>
</STOP_TIMER>
<REQUEST Name="startInvoiceTimer"/>
</ACTION>
</RULE>
</DEFINE_INIT>
<DEFINE_METHOD Name="startInvoiceTimer">
<RULE>
<ACTION>
<START_TIMER>
  <IDENTIFIED_BY>
    <NAME Value="INVOICE_TIMER"/>
  </IDENTIFIED_BY>
  <CALLBACK_DURATION Value="{duration(1,0,0,0)}"/>
  <CALLBACK_DATE Value="{incrDate(date
  ( getYear(),getMonth(),getDay() ),
  duration(0, 0, 0, (-1)) )}"/>
  <CALLBACK_ACTIONS>
    <REQUEST Name="startInvoiceRun">
      <START_TIME Value="{$callBackDateVar}"/>
    </REQUEST>
  </CALLBACK_ACTIONS>
</START_TIMER>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

# Guaranteed Messaging

A guaranteed message request is defined as follows in the rule. It has the following components.

- Guaranteed Request invocation (<GUARANTEED_REQUEST>)
  - Specifies the request to be of type guaranteed.
  - Specifies the API that needs to be invoked.
  - Specifies the service that this API is registered with.
- Header (<HEADER>)
  - Specifies the parameters that the Timer Service will use.
  - Specifies the mandatory message key, which can be any user defined string (MSG_KEY).
  - Specifies the number of retry attempts, default value being one (NUM_RETRIES).
  - Specifies the retry interval, default being zero (RETRY_INTERVAL).
  - Specifies the start date when the API is to be called first (START_DATE).
  - Specifies the result determining rule (RESULT_DETERMINER).
  - Specifies the guarantee level (High or Low. Default is High) (GUARANTEE_LEVEL).
- Body (<BODY>): Content to be sent to the API.
- On Success actions (<ON_SUCCESS>): Optional actions to be executed on a successful delivery.
- On Failure actions (<ON_FAILURE>): Optional actions to be executed on failure, after the specified number of retry attempts.

If guarantee level is set to “Low”, then the message delivery is attempted once before the guarantee request is persisted as a timer entry. So, the message delivery works in a semi-guaranteed fashion.

**Note:** Use semi-guaranteed messaging only if you are certain that you are willing to take the chance of losing a message.

*Example 1:*

Description: Example of setting up a guaranteed message. Once the message has been setup, the timer sink service will attempt delivery of this message. If the attempt is successful, it will execute the actions listed under ON_SUCCESS. If the delivery attempts fail, then the message is re-tried for a maximum of the specified number of retry times. If all attempts fail, then the ON_FAILURE actions are executed.

```
<DEFINE_METHOD Name="executeGuaranteedMsg">

<RULE>

<ACTION>

<GUARANTEED_REQUEST Name="callDummyExternalApi" ServiceName="DUMMY">

  <HEADER>
```

```
  <MSG_KEY Value="{$thisParam/ID/@Value}"/>
  <!-- Mandatory -->
  <NUM_RETRIES Value="{$thisParam/NUM_RETRIES
  /@Value}"/>
  <!--Default value = 1-->
  <RETRY_INTERVAL Value="{duration(0, 0, 0,
  $thisParam/RETRY_SECONDS/@Value)}"/>
  <!--default = 0-->
  <START_DATE Value="{incrDate(date(),
  duration(0,0,0,20))}"/>
  <!--Optional-->
  <RESULT_DETERMINER Rule="resultDeterminer"/>
  <!--Optional-->
  <GUARANTEE_LEVEL Value="High"/>
  <!--Optional-->
</HEADER>
<BODY>
  <ORDER Value="1">
    <LINE Value="{sum(1, 0)}"/>
    <LINE Value="{sum(1, 1)}"/>
  </ORDER>
  <ORDER Value="2">
    <LINE Value="{sum(2, 0)}"/>
    <LINE Value="{sum(2, 1)}"/>
  </ORDER>
</BODY>
<ON_SUCCESS>
  <PRINTLN Value="This is the thisDoc for msg
  {$thisDoc/@Name}"/>
  <PRINTLN Select="$thisDoc"/>
  <REQUEST Name="CallbackGMsgSuccess">
    <TO_XML DocVar="thisDoc"/>
```

```
      <ON Value="Success"/>
    </REQUEST>
  </ON_SUCCESS>
  <ON_FAILURE>
    <ADD_DEAD_LETTER>
      <TO_XML DocVar="thisDoc"/>
    </ADD_DEAD_LETTER>
    <REQUEST Name="CallbackGMsgFailure"
    ServiceName="DUMMY">
      <TO_XML DocVar="thisDoc"/>
      <ON Value="Failure"/>
    </REQUEST>
  </ON_FAILURE>
</GUARANTEED_REQUEST>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

## Success-failure Determination

When the message delivery is attempted, success is determined by the response from the invocation. An exception that occurs or an indication of failure is considered a failure.

Users can optionally specify a result determining rule. If this rule is specified, the rule is invoked right after an attempted delivery with the guaranteed message request data as parameter to this rule. The format of this input is in the next section. The rule then will have to indicate success or failure in the format specified below.

```
<!-- Attempt at delivery of guaranteed message-->
<REQUEST Name="callDummyExternalApi" ServiceName="DUMMY">
  <ORDER Value="1">
  <LINE Value="1.0"/>
  <LINE Value="2.0"/>
  </ORDER>
  <ORDER Value="2">
  <LINE Value="2.0"/>
  <LINE Value="3.0"/>
  </ORDER>
```

```
</REQUEST>
<!-- Success case -->
<RESPONSE Status="Success">
  <_RESULT Value="SUCCESS"/>
</RESPONSE>
<!-- Failure case -->
<RESPONSE Status="Success">
  <_RESULT Value="FAILURE"/>
</RESPONSE>
```

OR

```
<RESPONSE Status="Error">
  <SOME_ EXCEPTION_TRACE/>
</RESPONSE>
```

The on success and on failure actions will have access to variable thisDoc that will have the following.

The reply section contains the response from the API invoked.

```
<GUARANTEED_REQUEST Name="callDummyExternalApi" ServiceName="DUMMY">
  <HEADER>
     <IDENTIFIED_BY>
     <MSG_KEY Value="101010"/>
     <REQUEST_NAME Value="callDummyExternalApi"/>
     </IDENTIFIED_BY>
     <NUM_RETRIES Value="2"/>
     <RETRY_INTERVAL Value="65"/>
     <START_DATE Value="10/31/2003 16:41:42:143"/>
     <MSG_KEY Value="101010"/>
     <END_DATE Value="10/31/2003 16:43:52:143"/>
     <LAST_ATTEMPTED_DELIVERY_DATE Value="10/31/2003 16:41:48:921"/>
     <CURR_RETRY_COUNT Value="1"/>
  </HEADER>
  <BODY>
     <ORDER Value="1">
        <LINE Value="1.0"/>
        <LINE Value="2.0"/>
     </ORDER>
     <ORDER Value="2">
```

```
            <LINE Value="2.0"/>
            <LINE Value="3.0"/>
         </ORDER>
      </BODY>
      <REPLY>
         <RESPONSE Status="Success">
            <_RESULT Value="SUCCESS"/>
            <EXT_API_RESPONSE Value="SOME_DATA"/>
         </RESPONSE>
      </REPLY>
   </GUARANTEED_REQUEST>
```

# APPENDIX D Validation Services

### What's In This Appendix

This appendix provides information on Validation Service in Xcore.

### Related Topics

- Introduction
- Variables
- Attributes
- Using Validation Framework

# Introduction

Validation framework is used to do auto-fills and validations for an input XML document as per the validation spec specified. The result of the validation are returned to the caller after completing the full validation or after validating all the elements at the same level of the input xml document tree after encountering the first validation failure, whose severity is SEVERE_ERROR.

It is assumed that user of this documentation has knowledge about the following:

- XML X-Path expressions and Xcore X-Path expression function such as sum (), isValidValue(), count( ), getFirstDoc() etc
- Basic architecture of Distributed Order Management (x-core, services and their interactions) and access to the various services APIs

# Variables

The following are the predefined variables available for defining the validation specification:

- thisDoc. This variable is of type DocVar and points to the document for which the notification is called.

  Example: "`count($thisDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM) > 0`"
- thisField. Current field value.

  Example: "`count($thisDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM) > 0`"
- previousDoc. Previous document with the same name

  Example: "`$previousDoc/ID/@Value`"

- nextDoc. Next document with the same name

  Example: “`$nextDoc/ID/@Value`”
- previousDocList. List of previous documents

  Example: “`$previousDocList[ORDER_TYPE_ID/@Value='STANDARD']/ID/@Value`”
- nextDocList. List of documents to follow

  Example: “`$nextDocList[ORDER_TYPE_ID/@Value='STANDARD']/ID/@Value`”
- thisFieldDoc. Current field

  Example: ”`$thisFieldDoc/@OldValue != null`”
- thisParent. Current tag's parent tag.
  - Example1: "`$thisParent/ITEM_ID/@Value`"
  - Example2: "`$thisParent[name() = 'CUSTOMER_ORDER']/ORDER_TYPE_ID/@Value`"
- User defined variable. In addition to the above pre-defined variables, the user can define his own variables. Variables can be defined under the ‘`tag`’ tag. These variables can be defined in the following two ways:
  - Using an `Expression` attribute which is evaluated to get the value of the variable. In this case the value of the variable could be anything and the validation framework tries to create an appropriate object and stores it in the validation context.
  - Using the `x_rule` tag in the body of the variable. When you use `x_rule` to define a variable, you have access to thisReturn pre-defined variable. In the `x_rule` tag whatever you assign to thisReturn will finally get assigned to your variable.

    thisReturn can only be a well-formed xml in this case. Example of a variable defined using `x_rule` is shown below:

```
<VARIABLE Name="apply" IsNoCache="true">
<X_RULE>
  <APPEND_TO_XML SelectList="$thisReturn">
    <APPLY Value="true"/>
  </APPEND_TO_XML>
</X_RULE>
</VARIABLE>
```

# Attributes

The following attributes are available for various validation tags that are used to define the validation specification.

- Name: Name of the validationspec/variable/validation/autofill/field/tag depending on the tag in which this `Name` attribute is defined.

  For example, `<error_code Name="MISSING_REQUIRED_FIELD" Severity="SEVERE_ERROR"/>`

- Service Name: Name of the XService name. It is used to resolve validation name conflict.

  For example, `<user_validation Name="createPurchaseOrder" ServiceName="ORDER_ADMIN">`
- Severity: Severity of the error. The allowed values are one of, INFO, WARNING, ERROR, SEVERE_ERROR.

  For example, `<error_code Name="MIN_QTY_VIOLATION" Severity="ERROR"/>`
- IsFilter: The default value is false. if `IsFilter="true"` then expression used to compute variable should return boolean.

  For example, `<variable Name="shipModeDetails" ... IsFilter="true"/>`
- IsNoCache: The default value is false. Based on this flag this variable is computed every time it is referred or computed once when it is first time referred and then computed value is used when it is referred afterwards

  For example, `<variable Name="shipModeDetails" ... IsNoCache="true"/>`
- Expression: X-path expression to compute value of the variable/autofill/validation depending on the tag in which this attribute is defined.

  For example, `<validation Name="checkNonEmptyField" Expression="($thisField != null) ..../>`
- FilterExp: X-path expression to compute a filter.

  For example, `<validation Name="validate" FilterExp="$thisField != null" ..../>`
- FilterName: Variable name of type filter to compute filter.

  For example, `<validation Name="validate" filterName="isValidBillTo" ..../>`
- FilterExp/FilterName: If a filter is needed then the user is allowed to specify either FilterName or FilterExp. If both are specified then framework takes FilterExp.
- Error: error_code used when the validation fails.

  For example, `<validation Name="validate" .. Error="INVALID_VALUE"/>`
- IsActive: Default is false.

  For example, `<validation Name="validate" .. IsActive="yes" Error="INVALID_VALUE"/>`

## Types of Validations

There are three kinds of validation tags to specify the validation you want to perform. There is one for data type validation and one for mandatory validation. Although you can perform both of the above with a regular validation tag, it is recommended that you use the above for that purpose.

The following are the various types of validations and their syntax.

- data_type_validation: This validation can be used only for field validation. You can define the data type of a field as follows,

  `<field Name="CUM_RECEIVED_QTY" Type="float">`

This validation makes sure that the value of the field is of the same data type. You can also define resolutions which will be tagged along with the return XML document. For more information on resolutions see "Types of Resolutions". The following is an example of the usage of a data_type_validation.

*Example:*

```
<DATA_TYPE_VALIDATION FilterExp=""
Error="INVALID_CUM_RECEIVED_QTY" IsActive="yes">
<RESOLUTIONS>
<MANUAL_RESOLUTION>
   <RESOLUTION id="REJECT_DOCUMENT"/>
   <RESOLUTION id="MANUAL_DATA_ENTRY"/>
</MANUAL_RESOLUTION>
</RESOLUTIONS>
</DATA_TYPE_VALIDATION>
```

- mandatory_validation: This validation can be used only for field validation. This validation is to make sure that there is a value for this field. The following is an example of the usage of a mandatory_validation:

```
<MANDATORY_VALIDATION FilterExp="" Error="NULL_CUM_RECEIVED_QTY"
IsActive="yes">
<RESOLUTIONS>
<MANUAL_RESOLUTION>
   <RESOLUTION id="REJECT_DOCUMENT"/>
   <RESOLUTION id="MANUAL_DATA_ENTRY"/>
</MANUAL_RESOLUTION>
</RESOLUTIONS>
</MANDATORY_VALIDATION>
```

- validation: This validation can be used for field validations as well as tag validations. This kind of validation can be used to perform any other validation you may require other than the above two. For example, you may require that a value of the field is always '1'.

```
<VALIDATION Name="checkforOne" FilterName="All"
Expression="$thisField = '1'" IsActive="yes" Error="NOT_ONE"/>
```

- validation_group: Use the validation group tag when you want to group dependent validations into a group. For example, take the mandatory and data type validations above. If there is no value for the field there is no point executing the data type validation for that field. In this case, you would group mandatory and data type validation together, with the mandatory validation as the first. The data type validation is executed only if the mandatory validation succeeds. You can group as many validations as you like. The first failure of a validation stops the validation for that group.

  Some of the validations may not be as simple as the one above. There are two ways to handle this kind of validation, though there is no recommended approach:

  You can define a rule that does all the validation for you and make a call to that rule, which then returns either a SUCCESS or a ERROR.

You can use the x_rule tag available in the validation framework to code your validation logic in the validation specification. One important thing to remember if you use the x_rule approach is that the result of the validation has to be set to the property _RESULT on the thisReturn document. The following is an example of using the x_rule tag:

*Example:*

```
<VALIDATION Name="xyz" FilterName="All" IsActive="yes"
Error="adhdkh ">
<X_RULE>
<SET Var="buckets" FromValue="{$thisParent[name()='FORECAST']/
BUCKETS}"/>
<SET Var="bSpecEndDate"
FromValue="{$thisParent[name()='FORECAST']/END_DATE/@Value}"/>
<SET Var="invalidStartDate" FromValue="false"/>
<IF_TEST Test="date($thisField) &lt; date($bSpecEndDate)">
  <THEN>
    <IF_TEST Test="count($buckets/
    BUCKET[date(START_DATE/@Value)=date($thisField)])
    &lt;= 0">
    <THEN>
      <SET Var="invalidStartDate" FromValue="true"/>
    </THEN>
   </IF_TEST>
  </THEN>
</IF_TEST>
<IF_TEST Test="$invalidStartDate= 'true'">
  <THEN>
    <SET DocVar="thisReturn" Property="_RESULT"
    FromValue="ERROR"/>
  </THEN>
  <ELSE>
    <SET DocVar="thisReturn" Property="_RESULT"
    FromValue="SUCCESS"/>
  </ELSE>
</IF_TEST>
</X_RULE>
<RESOLUTIONS>
<MANUAL_RESOLUTION>
  <RESOLUTION id="REJECT_DOCUMENT"/>
  <RESOLUTION id="MANUAL_DATA_ENTRY"/>
</MANUAL_RESOLUTION>
```

```
</RESOLUTIONS>
</VALIDATION>
```

## Types of Resolutions

Resolutions can be used to fix validation errors like auto correcting the errors or for providing information to the caller about the resolutions that are possible. There are two kinds of resolutions:

- auto_resolution: This resolution has to have the Name attribute. The body of this resolution could be an X-rule.
- manual_resolution: The body of this resolution is appended to the error tag that is sent back to the caller. You can have x-path expressions in the body of this resolution. It is similar to the `TO_DOCVAR` tag in x-rules.

The following is an example of using the resolutions

*Example 1:*

You can define two auto resolutions and one manual resolutions shown below under the resolutions tag.

```
<RESOLUTIONS Apply="$apply/APPLY/@Value">
  <AUTO_RESOLUTION Name="AUTO_ACCEPT">
     <SET DocVar="thisDoc" Property="RESOLUTION"
     FromValue="NEW_ORDER_ACCEPT"/>
     <REQUEST Name="createNewOrder" ServiceName="FMX"
     AssignToVar="response">
        <TO_XML SelectList="$thisDoc"/>
        <SUPP_ORG_ID Value="{$thisParam/SUPP_ORG_ID/@Value}"/>
        <CUST_ORG_ID Value="{$thisParam/CUST_ORG_ID/@Value}"/>
        <ITEM_ID Value="{$thisParam/ITEM_ID/@Value}"/>
        <SHIP_TO_ID Value="{$thisParam/SHIP_TO_ID/@Value}"/>
     </REQUEST>
     <APPEND_TO_XML SelectList="$thisParam/ORDERS">
        <TO_XML Select="$response/ORDER"/>
     </APPEND_TO_XML>
  </AUTO_RESOLUTION>
  <AUTO_RESOLUTION Name="AUTO_REJECT">
     <SET DocVar="thisDoc" Property="RESOLUTION"
     FromValue="NEW_ORDER_REJECT"/>
     <SET DocVar="thisDoc" Property="STATUS" FromValue="REJECTED"/>
     <SET DocVar="thisDoc" Property="SCHEDULE_QTY" FromValue="0"/>
  </AUTO_RESOLUTION>
  <MANUAL_RESOLUTION>
```

```
      <CONTEXT>
         <TO_XML SelectList="$newReleaseRequestedContext/*"/>
      </CONTEXT>
      <TO_XML SelectList="$marButtons/*"/>
   </MANUAL_RESOLUTION>
</RESOLUTIONS>
```

If the validation fails, the resolutions are applied. The validation framework would evaluate the '`Apply`' attribute on the resolutions tag. It then tries to find the auto-resolution whose '`Name`' attribute value matches that of the '`Apply`' attribute. If it finds one, that resolution is applied. If it does not find a match and there is a manual resolution defined, it applies the manual resolution.

## Structure of a Validation Specification

Here is the high-level detail of the validation specification:

```
<USER_VALIDATIONS>
<USER_VALIDATION Name="..." ServiceName="..">
<ERROR_CODES>
<ERROR_CODE Name=".." Severity=".."/>
</ERROR_CODES>
<TAGS>
<TAG Name="..">
  <VARIABLES>
    <VARIABLE Name=".." IsFilter=".." Expression=".."
    IsNoCache=".."/>
  </VARIABLES>
  <VALIDATIONS>
    <VALIDATION Name=".." FilterExp=".." FilterName=
    ".." Expression=".." Error=".." IsActive="yes"/>
    <VALIDATION_GROUP Name="" IsActive="yes">
      <MANDATORY_VALIDATION FilterName="" FilterExp=""
      Error="" IsActive="yes">
        <RESOLUTIONS>
          <MANUAL_RESOLUTION>
          </MANUAL_RESOLUTION>
        </RESOLUTIONS>
```

```
        </MANDATORY_VALIDATION>
        <DATA_TYPE_VALIDATION FilterName=”” FilterExp=””
        Error="" IsActive="yes">
          <RESOLUTIONS>
            <MANUAL_RESOLUTION>
            </MANUAL_RESOLUTION>
          </RESOLUTIONS>
        </DATA_TYPE_VALIDATION>
      </VALIDATION_GROUP>
    </VALIDATIONS>
    <FIELDS>
      <FIELD Name="..">
        <AUTOFILLS>
          <AUTOFILL Name=".." FilterExp=".." FilterName=
          ".." Expression=".." IsActive="yes"/>
        </AUTOFILLS>
        <VALIDATIONS>
           <VALIDATION Name=".." FilterExp=".."
           FilterName=".." Expression=".."
           Error=".." IsActive="yes"/>
        </VALIDATIONS>
      </FIELD>
    </FIELDS>
</TAG>
</TAGS>
</USER_VALIDATION>
</USER_VALIDATIONS>
```

# Using Validation Framework

The following are the steps involved in using validation framework:

- Writing the Validation Specification
- Specifying the Validation Specification File

- Including Validation Specification File Path
- Invoking Validation Framework

## Writing the Validation Specification

You have to write validation specification file for your XML document to be validated and name the validation specification.

Typically validation specification name matches with the X-Rule name which is invoking the validation.

## Specifying the Validation Specification File

Include validation specification file in the appropriate xservice.xml so that validation service loads the validation specification to do the validation.

*Example:*

The following is an example from orderadmin service:

```
<DOM-CONFIG>
  <SERVICE-CONFIG Name="ORDER_ADMIN" TransactionCache="true">
     <REGISTER-HANDLERS>
        <HANDLER TagName="validationSpecFiles"
        ClassName="com.teradata.xservice.validation.ValidationSpecsH
        andler"/>
     </REGISTER-HANDLERS>
     <VALIDATIONSPECFILES>
        <VALIDATIONSPECFILE Name="xservice/orderadmin/xml/pce/
        validation/oaCreateOrderValidationBuy.xml"/>
        <VALIDATIONSPECFILE Name="xservice/orderadmin/xml/pce/
        validation/oaCreateBpoValidationBuy.xml"/>
     </VALIDATIONSPECFILES>
  </SERVICE-CONFIG>
</DOM-CONFIG>
```

## Including Validation Specification File Path

The validation specification file is always referred through the relative path. So we need to include the validation specification file path as part of the classpath, to read the validation specification correctly without an error.

## Invoking Validation Framework

You can invoke validation framework using `INVOKE_VALIDATION_FRAMEWORK`rule extension to validate the XML document as follows:

```
<INVOKE_VALIDATION_FRAMEWORK ServiceName="…" DocVar="…" ApiName="…"
AssignToVar="…"/>
```

Attributes:

- ServiceName.
- ApiName.
- DocVar.
- AssignToVar.

Example 1

```
<DEFINE_METHOD Name=" createOrderFromSOLineItem">
  <RULE>
     <ACTION>
        <SET Var="orderDoc" FromSelect="$thisParam/CUSTOMER_ORDER"/>
     </ACTION>
  </RULE>
<!-- Invoke the validation framework -->
  <RULE>
     <ACTION>
        <INVOKE_VALIDATION_FRAMEWORK ServiceName="ORDER_ADMIN"
        ApiName="createOrderFromSOLineItem"
        DocVar="orderDoc" AssignToVar="response"/>
        <IF_TEST Test="$ response/_RESULT/@Value != 'SUCCESS'">
          <THEN>
            <APPEND_CHILDREN DocVar="thisReturn"
            FromDocVar=" response"/>
            <EXIT/>
          </THEN>
          <ELSE>
            <SET Var="orderDoc" FromSelect="$response
            /CUSTOMER_ORDER"/>
          </ELSE>
        </IF_TEST>
     </ACTION>
  </RULE>
  <!-- ......  -->
</DEFINE_METHOD>
```

**Caution:** After calling validation framework make sure that you are updating the original XML document from validation framework response to get the autofilled value. See the succeeding example.

```
<INVOKE_VALIDATION_FRAMEWORK ServiceName="ORDER_ADMIN"
ApiName="createOrderFromSOLineItem"
```

```
DocVar="orderDoc" AssignToVar="response"/>
<SET Var="orderDoc" FromSelect="$response/CUSTOMER_ORDER"/>
```

Input:

Sample input for createPurchaseOrder API:

```
<CUSTOMER_ORDER>
  <DOC_TYPE Value="ORDER"/>
  <ORDER_TYPE_ID Value="PURCHASE_ORDER" Editable="no"/>
  <ORDER_ORIGIN_ID Value="BUYER"/>
  <CUSTOMER_ORG_ID Value="ORG_10"/>
  <ORDER_POINT_ID Value="OP_1"/>
  <SUPPLIER_ID Value="ORG_1"/>
  <SHIP_MODE Value="GROUND"/>
  <SHIP_TO_ID Value="SP_1"/>
  <TAXABLE Value="true" Editable="yes"/>
  <SELLING_ORG_ID Value="ORG_1"/>
  <ORDER_LINE_ITEMS>
     <ORDER_LINE_ITEM>
     <LINE_ITEM_TYPE Value="REGULAR"/>
     <ITEM_ID Value="151-S"/>
     <ITEM_DESCR Value="BORESCOPE SET"/>
     <UNIT_PRICE Value="500"/>
     <LIST_PRICE Value="9471.9"/>
     <ORDERED_QTY Value="6"/>
     <QTY_PER Value="6"/>
     <UOM Value="EA"/>
     <ORDER_REQUESTS>
       <ORDER_REQUEST>
         <DELIVERY_DATE Value="01/25/2003"/>
         <UOM Value="EA"/>
         <REQ_QTY Value="6"/>
       </ORDER_REQUEST>
     </ORDER_REQUESTS>
     <EXT_REF_ID Value="1"/>
     </ORDER_LINE_ITEM>
  </ORDER_LINE_ITEMS>
</CUSTOMER_ORDER>
```

Output:

Sample success output for createPurchaseOrder API:

```
<RESPONSE Status="Success">
  <_RESULT Value="SUCCESS"/>
  <CUSTOMER_ORDER>
     <DOC_TYPE Value="ORDER"/>
     <ORDER_TYPE_ID Value="PURCHASE_ORDER" Editable="no"/>
     ....
     <ORDER_LINE_ITEMS>
     <ORDER_LINE_ITEM>
       <LINE_ITEM_TYPE Value="REGULAR"/>
       <ITEM_ID Value="151-S"/>
       ....
       <ORDER_REQUESTS>
         <ORDER_REQUEST>
           <DELIVERY_DATE Value="01/25/2003"/>
           <REQ_QTY Value="6"/>
           <UNIT_PRICE Value="500"/>
           ....
           <ID Value="REQ-206"/>
         </ORDER_REQUEST>
       </ORDER_REQUESTS>
       ...
       <ID Value="LI-206"/>
       <CREATION_DATE Value="02/11/2003 10:56:47:066"/>
     </ORDER_LINE_ITEM>
     </ORDER_LINE_ITEMS>
  ...
  <ID Value="PO-206"/>
  <CREATION_DATE Value="02/11/2003 10:56:47:066"/>
  </CUSTOMER_ORDER>
</RESPONSE>
```

Error Output:

Sample error output for createPurchaseOrder api:

```
<RESPONSE Status="Success">
  <_RESULT Value="ERROR" Details="true">
  <_ERROR Value="INVALID_VALUE" Severity="ERROR"/>
  </_RESULT>
```

```
<CUSTOMER_ORDER>
   <DOC_TYPE Value="ORDER"/>
   <ORDER_TYPE_ID Value="PURCHASE_ORDER" Editable="no"/>
   ....
   <ORDER_LINE_ITEMS>
      <ORDER_LINE_ITEM>
        <LINE_ITEM_TYPE Value="REGULAR"/>
        <ITEM_ID Value="151-S"/>
        <ITEM_DESCR Value="BORESCOPE SET"/>
        <ORDERED_QTY Value="6"/>
        <QTY_PER Value="6"/>
        <UOM Value="EA"/>
        <ORDER_REQUESTS>
          <ORDER_REQUEST>
            <DELIVERY_DATE Value="01/25/2003"/>
            ...
          </ORDER_REQUEST>
        </ORDER_REQUESTS>
        <EXT_REF_ID Value="1"/>
        <ORDER_POINT_ID Value="OP_1"/>
        ....
        <UNIT_PRICE>
          <_ERRORS Value="ERROR">
            <_ERROR Value="INVALID_VALUE" Severity="ERROR"/>
          </_ERRORS>
        </UNIT_PRICE>
        <_ERRORS Value="ERROR"/>
      </ORDER_LINE_ITEM>
   <_ERRORS Value="ERROR"/>
   </ORDER_LINE_ITEMS>
<BILL_TO_ID Value="BT_1"/>
<INVOICE_METHOD Value="SHIPMENT"/>
.....
<ORDER_SUB_TYPE Value="NORMAL"/>
<_ERRORS Value="ERROR"/>
</CUSTOMER_ORDER>
</RESPONSE>
```

# APPENDIX E Upload Services

### What's In This Appendix

This appendix provides information about Upload Services.

Topics include:


- Introduction
- Assisted Upload
- unassisted Upload


# Introduction

The requirement of the data upload service is to upload various kinds of data files into the MDM application. The upload can be performed using the UI or unassisted from a server location.

# Assisted Upload

Data upload service supports both the xml and csv format.

## XML Based Data Upload Template

A template identifies a specific type or group of entity transactions. A template is created using the following Request.

```
<REQUEST Name="addTemplate">

  <TEMPLATE Name="<template_name>" DisplayName="<display_name>"
  TemplateType="Transaction.<template_name>" DataType="XML"
  ServiceName="E2E" ApiName="<api_name>" UnassistedUpload="false"
  NumRetries="5" RetryFreq="2" UserData="excelUpload"
  MiscData="<entity_logical_name>">

</REQUEST>
```

Attributes:

- DisplayName: Display name of the template on the UI.
- Name: Unique name referenced in the data file and also used while creating the source sub-folders.
- ApiName: Name of the API to invoke.

- TemplateType: Transaction based template.
- DataType: XML
- ServiceName: Target service name.
- UserData: “ExcelUpload”
- NumRetries: Number of upload retries in case of failure.
- RetryFreq: Time interval between each retry.
- OnCompleteApiName: API to invoke upon completion of upload.
- UnassistedUpload: Boolean value to turn unassisted upload on or off
- InputFilePrefix: Prefix (with a mandatory “.” in it) to the data file
- MiscData: Logical name of the entity.

The following example shows the `Entity_AttributeSet` template:

```
<REQUEST Name="addTemplate">

<TEMPLATE Name="EntityAttributeSet" DisplayName="EntityAttributeSet"
TemplateType="Transaction.EntityAttributeSet" DataType="XML"

ServiceName="E2E" ApiName="EntityAttributeSet"
UnassistedUpload="false" NumRetries="5" RetryFreq="2"
UserData="excelUpload" MiscData="entity_attributeset"

<PROPERTY Name="attributeSetID" />

<PROPERTY Name="updatedLogicalTableName" />

<PROPERTY Name="updatedPhysicalTableName" />

<PROPERTY Name="description" />

<PROPERTY Name="name" />

<PROPERTY Name="SYS_TARGET_ID" />

<PROPERTY Name="SYS_AUTH_ID" />

<PROPERTY Name="SOURCE" />

<PROPERTY Name="SYS_TYPE" />

</TEMPLATE>

</REQUEST>
```

# unassisted Upload

Identify a “root” folder underneath which you want to create all the process folders. The absolute path to this root folder is configured in a property file as given below.

- Filename

  *<MDM_Install_Directory>/cfg/properties/dataupload.xml*
- Parameter

  `<param Name="UnassistedUploadBase"`
- Value

```
<absolute path to the root folder>
```

**Note:** The root folder and the process folders can be maintained outside the MDM directory structure.

The process folders are:

- Source
- In-Process
- Archive
- Error
- Reprocess

Unassisted upload requires the data file to be available in a particular location. This location is: `/<root folder>/source/<archive>`. The upload service is prompted through a poster script. An example script to invoke the upload is as follows:

```
<REQUEST Name="unassistedUploadTimerHandler">
  <TIMER_CALLTIME Value="01/08/2004 16:20:00"/>
  <REQUEST Name="startUploadTimer">
    <UPLOAD_TEMPLATE ExistingDocument="yes">
      <NAME Value="EntityAttributeSet"/>
    </UPLOAD_TEMPLATE>
    <CURRENT_TIME Value="01/08/2004 16:20:00"/>
  </REQUEST>
</REQUEST>
```

It is a good practice to create a separate script file for every template.

# APPENDIX F ID Generation

### What's In This Appendix

This appendix provides information about Id generation.

### Related Topics

- Introduction
- TimeIdGeneration
- TableIdGeneration
- BlockIdGeneration
- Configuration

# Introduction

An identifier is used to identify an entity in Database. For example, an Order Id is used to identify an Order from rest of the Orders in the System. MDM supports configurable way to generate Id for the entities in System. This should be configured in the service descriptor.

MDM supports the following three methods of Id generation.

- TimeIdGeneration.
- TableIdGeneration.
- BlockIdGeneration.

# TimeIdGeneration

This is the default Id generation used in MDM. It generates the Id based on current system time and the entity for which the Id is being generated. This method is used when you enable the Surrogate Key on a table (For more information refer to the *Data Modeling chapter* in the *MDM Platform Studio User Guide*).

# TableIdGeneration

In this scheme MDM maintains a table called `ID_SQR`. The schema for `ID_SQR` Table is defined as follows:

```
<DOCUMENT Name="ID_SEQUENCE" TableName="ID_SQR">
```

```
<PROPERTY Name="DOC_NAME" Type="string(64)" ColumnName="DOC_NAME"
PrimaryKey="yes"/>
<PROPERTY Name="SEQUENCE_NUMBER" Type="int" ColumnName="SEQ_NR" />
</DOCUMENT>
```

`ID_SQR` table keeps the next Id for each entity. This table is common for all the co-located X-Service(s).

# BlockIdGeneration

The TableIdGenerator makes a call to the Database for generating every Id. To avoid this, MDM supports another method for generating IDs called the BlockIDGeneration.

In the BlockIDGenerator, system fetches blocks of Id from the Database and keeps it in the memory. When the block is served out it goes to the Database to fetch next block. The block size is configured in the IdGen spec of the service descriptor.

# Configuration

When using any of the three methods of generating IDs—Table Id, Sequence Id or Block Id, you have to first configure the X-Service file by setting the 'idgen' block with details of the document, prefix, delimiter, sequenceName, block size (if applicable), and the handler for each document for which Id generation is automated.

```
<idgen>
<document Name="CHANGE_ORDER_NOTES" prefix="" delimiter=""
handler="com.teradata.xcore.idgen.TimeIdGenerator" />
<document Name="USR_SECURITY_AUDIT_ID" prefix="Id-
sequenceName="USR_SECURITY_AUDIT_ID_SEQ"
handler="com.teradata.xcore.idgen.BlockTableIdGenerator"blockSize="1
00"/>
</idgen>
```

**Note:** It is recommended to use decimal or double data type for large value or for primary key in table column. Float data type must be avoided.

The following attributes apply to all Id generation schemas:

Attributes of `<document>`

- document: Name of document for which sequence is generated.
- name: It is the name of the document. The user has to configure the Id generation scheme for each Document that are defined in the Service.
- prefix: It identifies the prefix to be used in Id.

  For example, `CO` for `CUSTOMER_ORDER` Document.

- delimiter: It specifies the character to be used after prefix.
- blockSize: It identifies the INCREMENT size of the Sequence in the Database.
- handler: The handler attribute specifies the Id generation scheme to be used for a Document. The following are the valid values of the handler:
  - `com.teradata.xcore.idgen.TimeIdGenerator`
  - `com.teradata.xcore.idgen.BlockTableIdGenerator`

## Generating Next Id

After completing the configuration, you can invoke the below code to generate next Id.

```
<CREATE_DOCUMENT_ID NAME="Name"AssignToVar="NextId"/>
```

For example, to generate Id USER_SECURITY_AUDIT_ID, invoke:

```
<CREATE_DOCUMENT_ID NAME="USR_SECURITY_AUDIT_ID"AssignToVar="auditId"/>
```

Variable “auditId” contains next generated Id value with “Id-”. Id block of 100 will be retrieved and will be kept in cache to avoid frequent call to DB for next Id.

# APPENDIX G Events and Request Interfaces

### What's In This Appendix

This appendix provides information on X-Document.

### Related Topics

- Defining an Event
- Subscribing Events
- Raising Events
- Defining a Request Interface

## Defining an Event

Events are part of an X-service. An event name is unique within a service. Events can be associated with a service by including the event definition file as part of the service config file. Typically event definition is specified in the service that raises the event.

The relevant portions of the service config file are shown below:

```
<dom-config>
   <service-config Name="FMX" TransactionCache="true"
   Register="true">
   <eventdefnfiles>
      <eventdefnfile Name="xservice/fmx/xml/eventdef/myevents.xml"/>
   </eventdefnfiles>
   </service-config>
</dom-config>
```

The above configuration is done automatically by MDM Studio when you attach an event definition file to the X-Service in your project.

Attributes:

An event definition (`<EVENT_DESCRIPTOR>`) consists of the following:

- Name. The fully qualified name of the event is `//<service-name>/<event-name>`. This name is globally unique.
- Category of the Event
- Signature of the event. The signature is expressed through `EVENT_DOC` tags
- Workflow keys for the event. This is optional.

- `thisParam` is a predefined variable referencing the `<RAISE_EVENT>` document.

*Example 1:*

```
<event_descriptor Name="newForecastArrived" Category="FORECAST">
  <EVENT_DOC>
     <RAISE_EVENT Name="newForecastArrived">
        <FCST_LINE_ID Value="xxx"/>
     <RAISE_EVENT/>
  </EVENT_DOC>
  <KEYS>
     <KEY Value="$thisParam/FCST_LINE_ID/@Value"/>
  </KEYS>
</event_descriptor>
```

*Example 2:*

You can define one or more events in an event definition file (myevents.xml) as shown in the following example:

```
<DOCUMENTS>
  <event_descriptor Name="newForecastArrived" Category="FORECAST">
     <EVENT_DOC>
     <RAISE_EVENT Name="newForecastArrived">
       <FCST_LINE_ID Value="xxx"/>
     <RAISE_EVENT/>
     </EVENT_DOC>
     <KEYS>
        <KEY Value="$thisParam/FCST_LINE_ID/@Value"/>
     </KEYS>
  </event_descriptor>
  <event_descriptor Name="ForecastLineValidationFailed"
  IsRequest="false">
     <EVENT_DOC>
     <RAISE_EVENT Name="ForecastLineValidationFailed">
       <EVENT_HEADER>
         <DOC_ID Value="xxx"/>
         <REF_DOC_ID Value="xxx"/>
         <REF_DOC_TYPE Value="PROBLEM_LINE[Type=Constant]
         "/>
         <SELLING_ORG_ID Value="xxx"/>
         <CUSTOMER_ORG_ID Value="xxx"/>
```

```
            <ITEM_ID Value="xxx"/>
            <LOCATION_ID Value="xxx"/>
          </EVENT_HEADER>
       </RAISE_EVENT>
    </EVENT_DOC>
</event_descriptor>
</DOCUMENTS>
```

# Subscribing Events

The following are different ways in which you can subscribe to an event:

- Defining a Listener in X-Rules
- Defining a Listener in Workflow
- Defining a Listener in Through Event Spec

## Defining a Listener in X-Rules

Description: You can use the `DEFINE_LISTENER` type of business request to define an event listener. Whenever the event occurs, the corresponding X-Rule(s) would be executed.

Syntax:

<DEFINE_LISTENER Name = "Listener_Name" Descriptor=" ">

Attributes:

- Name: Name of the listener.
- Descriptor: Event Descriptor (defined in the event definition file).

*Example 1:*

```
<DEFINE_LISTENER Name="newForecastArrivedListener1" Descriptor="//
FMX/newForecastArrived">
   <RULE>
      <ACTION>
         <PRINTLN Value="The event key = {$thisParam/FCST_LINE_ID/
         @Value}"/>
      </ACTION>
   </RULE>
</DEFINE_LISTENER>
```

## Defining a Listener in Workflow

You can define a listener through an event node in workflows.

Syntax:

```
<EVENT Name="WaitForNewForecastArrivedEvent" Description="Wait for
new Forecast Arrival" InputVar="{lineForecast,lineForm}"
Descriptor="//FMX/newForecastArrived">
```

Attributes:

- Name: Name of the listener.
- Descriptor: Event Descriptor (defined in the event definition file).
- TransformRule: Transform the rule to EMx format. It is required if the signature of the event is different from the format required by EMx.

*Example 1:*

```
<EVENT Name="WaitForNewForecastArrivedEvent" Description="Wait for
new Forecast Arrival" InputVar="{lineForecast,lineForm}"
Descriptor="//FMX/newForecastArrived">
  <PRE_ACTIONS/>
  <POST_ACTIONS>
     <ACTION Name="Print Info">
        <PRINTLN Value="The event key = {$thisParam/FCST_LINE_ID/
        @Value}"/>
     </ACTION>
  </POST_ACTIONS>
  <KEYS>
     <KEY Value="$fcLineId"/>
  </KEYS>
  <NEXT_NODES>
     <NEXT_NODE Name="Done" Description=""/>
  </NEXT_NODES>
</EVENT>
```

## Defining a Listener in Through Event Spec

You can register an Emx event spec as a listener for an event by specifying the event descriptor as part of the Emx event spec definition.

*Example 1:*

```
<EVENTSPEC>
  <EVENT Name="FORECAST.ForecastLineValidationFailed" Descriptor="//
  FMX/ForecastLineValidationFailed" TransformRule="//FMX/
  ForecastLineValidationFailedTransform" >
  [other Emx spec attributes]
  </EVENT>
```

```
</EVENTSPEC>
```

You can optionally specify a transform rule through TransformRule attribute on the event spec to transform the rule to X-Service format. This is required if the signature of the event is different from the format required by X-Service.

*Example 2:*

Following is an example of the transform rule defined in FMX service. This transforms the event format to the standard EMx event format. This rule spits the output as:

```
<DEFINE_METHOD Name="ForecastLineValidationFailedTransform">
<API_DOC>
 <INPUT>
<REQUEST Name="ForecastLineValidationFailedTransform"
OrderIndicator="true" Any="true">
<RAISE_EVENT Name="ForecastLineValidationFailed" Repeatable="true">
  <EVENT_HEADER>
    <DOC_ID Value="xxx"/>
    <REF_DOC_ID Value="xxx"/>
    <REF_DOC_TYPE Value="PROBLEM_LINE[Type=Constant]
    "/>
    <SELLING_ORG_ID Value="xxx"/>
    <CUSTOMER_ORG_ID Value="xxx"/>
    <ITEM_ID Value="xxx"/>
    <LOCATION_ID Value="xxx"/>
  </EVENT_HEADER>
</RAISE_EVENT>
</REQUEST>
 </INPUT>
 <OUTPUT>
<RESPONSE OrderIndicator="true" Any="true">
<EVENT Repeatable="true">
  <HEADER>
    <ANY/>
  </HEADER>
  <DETAILS>
    <ANY/>
```

```
  </DETAILS>
</EVENT>
</RESPONSE>
 </OUTPUT>
</API_DOC>
<RULE>
<ACTION>
<PRINT Value="Invoked add Forecast line Validation Failed Event"
Nl="yes"/>
<FOR_EACH SelectList="$thisParam/RAISE_EVENT"
CurrentElement="eventDoc">
  <APPEND_TO_XML SelectList="$thisReturn">
    <EVENT>
      <HEADER>
        <DOC_ID Value="{$eventDoc/EVENT_HEADER
        /DOC_ID/@Value}"/>
        <DOC_TYPE Value="FORECAST_LINE"/>
        <REF_DOC_ID Value="{$eventDoc/EVENT_HEADER
        /REF_DOC_ID/@Value}"/>
        <REF_DOC_TYPE Value="{$eventDoc/EVENT_HEADER
        /REF_DOC_TYPE/@Value}"/>
        <SELLING_ORG_ID Value="{$eventDoc
        /EVENT_HEADER/SELLING_ORG_ID/@Value}"/>
        <CUSTOMER_ORG_ID Value="{$eventDoc
        /EVENT_HEADER/CUSTOMER_ORG_ID/@Value}"/>
        <SELLING_ORG_NAME Value="ABC"/>
        <CUSTOMER_ORG_NAME Value="DEF"/>
        <ITEM_ID Value="{$eventDoc/EVENT_HEADER
        /ITEM_ID/@Value}"/>
        <LOCATION_ID Value="{$eventDoc/EVENT_HEADER
        /LOCATION_ID/@Value}"/>
        <LOCATION_NAME Value="XYZ"/>
        <DESCRIPTION Value="Forecast Validation
```

```
        Failed"/>
      </HEADER>
    </EVENT>
  </APPEND_TO_XML>
</FOR_EACH>
</ACTION>
</RULE>
</DEFINE_METHOD>
```

The job of transform rule is to collect any additional details required by EMx and format it as per EMx requirements. In the preceding example, it is providing additional information like: `SELLING_ORG_NAME` and `CUSTOMER_ORG_NAME`.

The intent is for all services to use one common mechanism (`<RAISE_EVENT>`) to raise events.

The `<CALL_EVENT>` tag will be deprecated and eventually replaced by `<RAISE_EVENT>`.

# Raising Events

Description: Events can be raised in the system by issuing an `<RAISE_EVENT>` command.

Syntax:

```
<RAISE_EVENT Name="Event_name">
```

*Example 1:*

```
<RAISE_EVENT Name="ForecastLineValidationFailed">
  <EVENT_HEADER>
     <DOC_ID Value="FC-LI-201"/>
     <REF_DOC_ID Value="REQ-PROB-101"/>
     <REF_DOC_TYPE Value="PROBLEM_LINE"/>
     <SELLING_ORG_ID Value="1"/>
     <CUSTOMER_ORG_ID Value="2"/>
     <ITEM_ID Value="SUP-MOT-ITEM1"/>
     <LOCATION_ID Value="SUP-MOT-SHIPTO1"/>
  </EVENT_HEADER>
</RAISE_EVENT>
```

The body of `RAISE_EVENT` should match the `EVENT_DOC` specified in the event definition file.

The following is an example of the response to the caller of the `<RAISE_EVENT>` command:

```
<RESPONSE>
  <NUMBER_OF_LISTENERS Value="5"/>
</RESPONSE>
```

Attributes:

- Name: It is the name of the service. It is optional. If service name is not provided then the service is assumed to be the service of the caller.

An event can have multiple listeners. All of the listeners are notified of the event asynchronously. Each listener will be notified independently in a separate transaction.

You can also raise multiple events in a single call as follows:

```
<RAISE_EVENTS ServiceName="FMX">
  <RAISE_EVENT Name="ForecastLineValidationFailed">
     <EVENT_HEADER>
        <DOC_ID Value="FC-LI-201"/>
        <REF_DOC_ID Value="REQ-PROB-101"/>
        <REF_DOC_TYPE Value="PROBLEM_LINE"/>
        <SELLING_ORG_ID Value="1"/>
        <CUSTOMER_ORG_ID Value="2"/>
        <ITEM_ID Value="SUP-MOT-ITEM1"/>
        <LOCATION_ID Value="SUP-MOT-SHIPTO1"/>
     </EVENT_HEADER>
  </RAISE_EVENT>
  <RAISE_EVENT Name="ForecastLineValidationFailed">
     <EVENT_HEADER>
        <DOC_ID Value="FC-LI-209"/>
        <REF_DOC_ID Value="REQ-PROB-103"/>
        <REF_DOC_TYPE Value="PROBLEM_LINE"/>
        <SELLING_ORG_ID Value="1"/>
        <CUSTOMER_ORG_ID Value="2/>
        <ITEM_ID Value="SUP-MOT-ITEM1"/>
        <LOCATION_ID Value="SUP-MOT-SHIPTO1"/>
     </EVENT_HEADER>
  </RAISE_EVENT>
  <RAISE_EVENT Name=" MissingFirmQualifier ">
     <EVENT_HEADER>
        <DOC_ID Value="FC-LI-209"/>
        <REF_DOC_ID Value="REQ-PROB-103"/>
        <REF_DOC_TYPE Value="PROBLEM_LINE"/>
```

```
        <SELLING_ORG_ID Value="1"/>
        <CUSTOMER_ORG_ID Value="2"/>
        <ITEM_ID Value="SUP-MOT-ITEM1"/>
        <LOCATION_ID Value="SUP-MOT-SHIPTO1"/>
      </EVENT_HEADER>
  </RAISE_EVENT>
</RAISE_EVENTS>
```

# Defining a Request Interface

Description: It is possible to define the signature of a REQUEST without providing a concrete implementation through `<request_descriptor>` tag. The implementation can be provided by subscribing to the REQUEST as a listener. This feature is only necessary if a REQUEST is to be implemented through workflow event or start nodes. Otherwise you can directly provide a concrete implementation of the REQUEST through `<DEFINE_METHOD>` in X-Rules.

Syntax:

```
<request_descriptor Name="" Category>
```

Attributes:

- Name: Descriptor Name.
- Category: Category name (for classification only – optional).
- Access: The valid values are Public, Protected and Private. Default=Protected.
- Signature: It is the signature of the request. The signature is express through API_DOC tags.
- Keys: Workflow Keys for the request. This is optional.

*Example 1:*

```
<request_descriptor Name="receiveApproveEvent" Access="Public"
Category="Fcst Analysis" DefaultHandler="fcstApprovalErrorHandler">
<API_DOC>
<INPUT>
<REQUEST Name="receiveApproveEvent">
  <FCST_LINE_ID Value="xxx[Type=string]"/>
</REQUEST>
</INPUT>
<OUTPUT>
<ON_SUCCESS>
```

```
  <RESPONSE>
    <_RESULT Value="SUCCESS|ERROR">
      <ERROR_CODE Value="xxx" Optional="true"/>
    </_RESULT>
  </RESPONSE>
</ON_SUCCESS>
</OUTPUT>
</API_DOC>
  <KEYS>
     <KEY Value="$thisParam/FCST_LINE_ID/@Value"/>
  </KEYS>
</request_descriptor>
```

Additional information:

- You can register a listener to the Request Interface through X-Rules or Workflow Event or Start nodes. It does NOT make sense to have an EMx event spec corresponding to a request interface.
- You can only have one listener for a request interface. The last listener will replace any of the previous ones. The listener for a request interface will be invoked in the same transaction as the caller who invoked the `<REQUEST>`.
- The listener is responsible for generating the response in accordance with the output signature defined in the `API_DOC`.
- You can also specify a DefaultHandler. The default handler will be invoked in case there is no listener registered for the request interface.

*Example 2:*

```
<DEFINE_METHOD Name="fcstApprovalErrorHandler">
  <ACTION>
     <APPEND_TO_XML SelectList="$thisREturn">
        <_RESULT Value="ERROR">
          <ERROR_CODE Value="INVALID_MESSAGE"/>
        </_RESULT>
     </APPEND_TO_XML>
  </ACTION>
</DEFINE_METHOD>
```

The request interfaces can be defined in the event definition file along with events.

# APPENDIX H Using Rules Language Project

### What's In This Appendix

This appendix walks you through the usage of the Rules language project.

Topics include:


- Location
- Generating Schema
- Rules


## Location

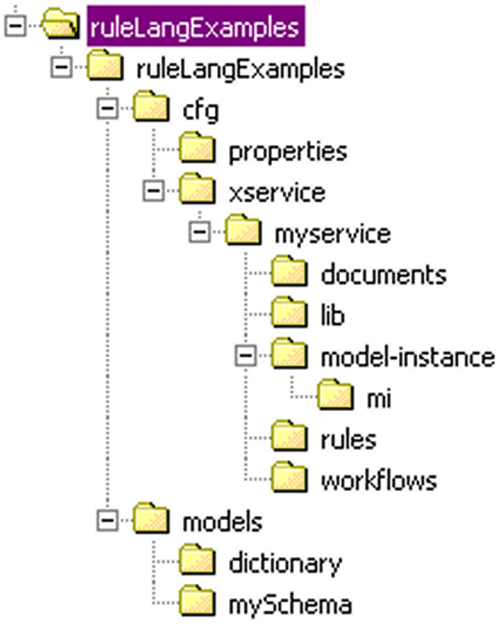
If you have successfully installed MDM, you should be able to find the Rules language project as one of the sample projects with the installation.

The following is the location where you can find the rules language project in your MDM installation:

```
<STUDIO_HOME>\services\studio\samples\ruleLangExamples\
```

The following is the directory structure of the ruleLangExamples project:

Figure 97: Directory Structure



The following are the files in the ruleLangExamples directory:

Table 99: Files in the ruleLangExamples Directory

| File | Directory |
|---|---|
| xserver.xml | ruleLangExamples\ruleLangExamples\cfg\properties |
| myservice.xml | ruleLangExamples\ruleLangExamples\cfg\properties |
| midocs.xml | ruleLangExamples\ruleLangExamples\cfg\xservice\myservice\documents |
| mi.xml | \ruleLangExamples\ruleLangExamples\cfg\xservice\myservice\model-instance |
| mySchema.xml | \ruleLangExamples\ruleLangExamples\models |
| myDictionary.xml | ruleLangExamples\ruleLangExamples\models\dictionary |
| ruleLangExamples.owb | ruleLangExamples |
| TestInstance.class | \ruleLangExamples\ruleLangExamples\cfg\xservice\myservice\lib |

All the Rule files and Workflow files that are used in the project are available in the rules and workflows directory at `ruleLangExamples\ruleLangExamples\cfg\xservice\myservice`.

# Generating Schema

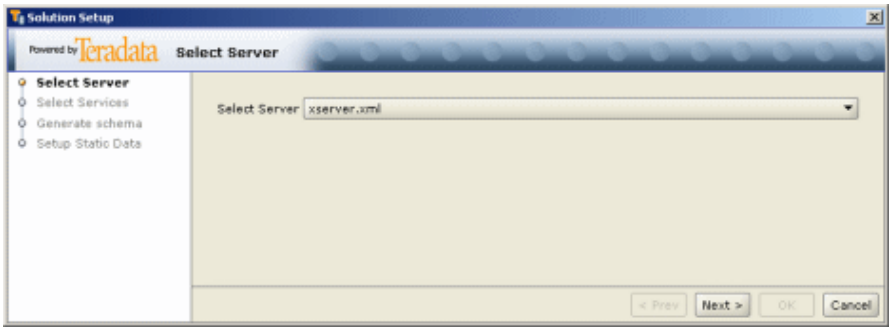
You have to generate schema in order to run the ruleLangExamples.owb project. Perform the following steps to generate the schema:

To generate schema:

1. Open the ruleLangExamples.owb project file.
2. Change the database configurations.
3. Right click on the **Solution** folder and click **Solution Setup**.

   The **Solution Setup** wizard is displayed.

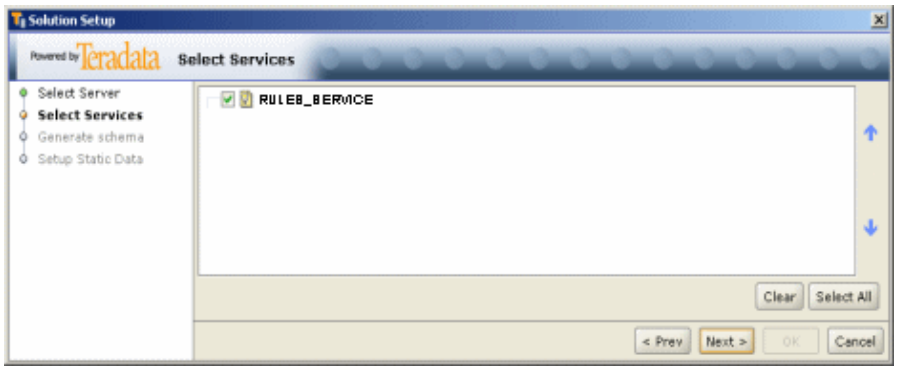
Figure 98: Solution Setup




4 Select **xserver.xml** and click **Next**.

The **Select Services** screen is displayed.

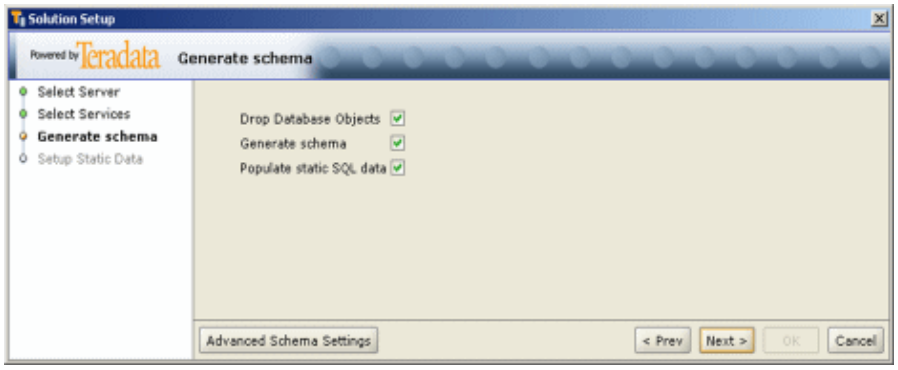
Figure 99: Select Services




5 Select **RULES_SERVICE** and click **Next**.

The **Generate Schema** screen is displayed.

Figure 100: Generate Schema



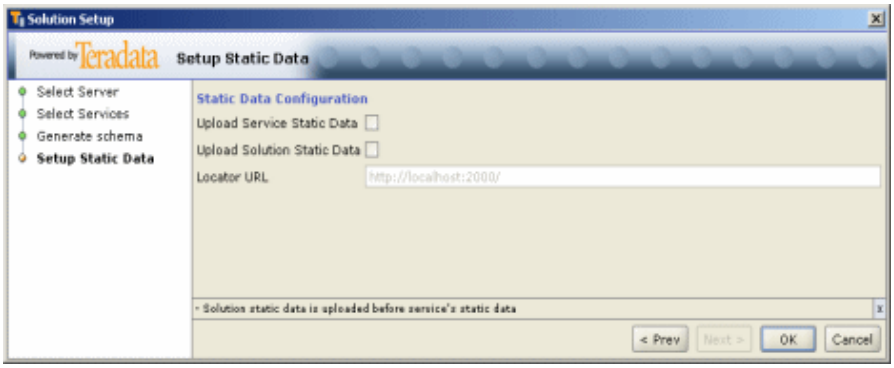

6 Select the **Drop Database Objects**, **Generate Schema**, and **Execute Service SQL** options.

7 Click **Advanced Schema Settings** and change the Maximum Table Length Name to *24*.

8 Click **Next**.

The **Setup Static Data** screen is displayed.

Figure 101: Setup Static Data




9 Do not select any values. Click **OK** to finish the schema generation process.

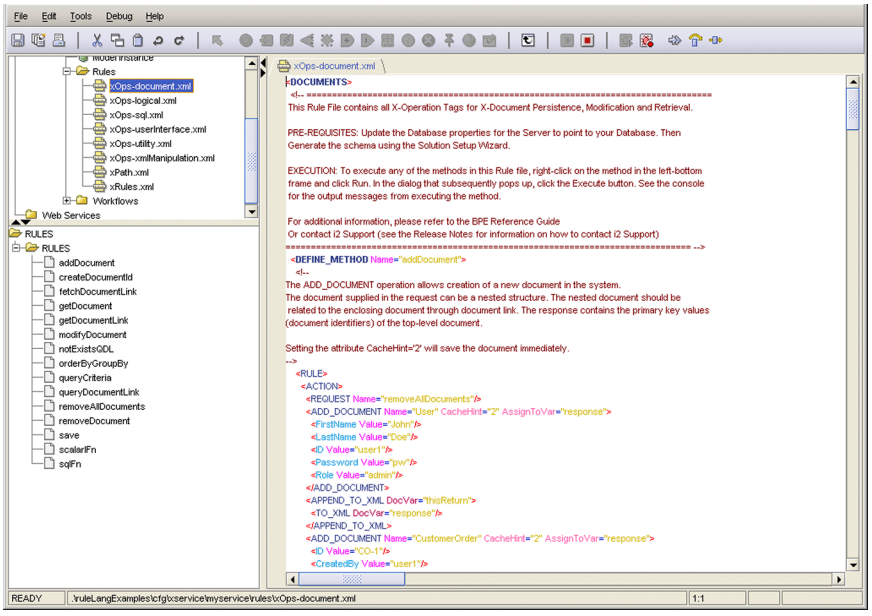
The **Successfully generated schema** message is displayed in the console if the schema is generated successfully

# Rules

Each Rule file in the left navigation pane under the Rules folder corresponds to a set of Operations/Rules/X-Path functions, described earlier in this guide. Each file contains a set of methods (or rules), each method corresponding to an Operation or a Rule or an X-Path function. When you double click on a rule file, the operation/rules contained in the file are displayed in bottom section of the left navigation pane.

For example, double click on xOps-document.xml. The operation/rules are displayed in the bottom section of the left navigation pane.

Figure 102: Operation/Rule



Double click on the operation/rule to view its definition in the right pane; examine the syntax of the operation/rule carefully to understand its usage. The comments in the operation/rule helps you understand what the rule does.

## Running Rules

Perform the following steps to execute the rules in the project.

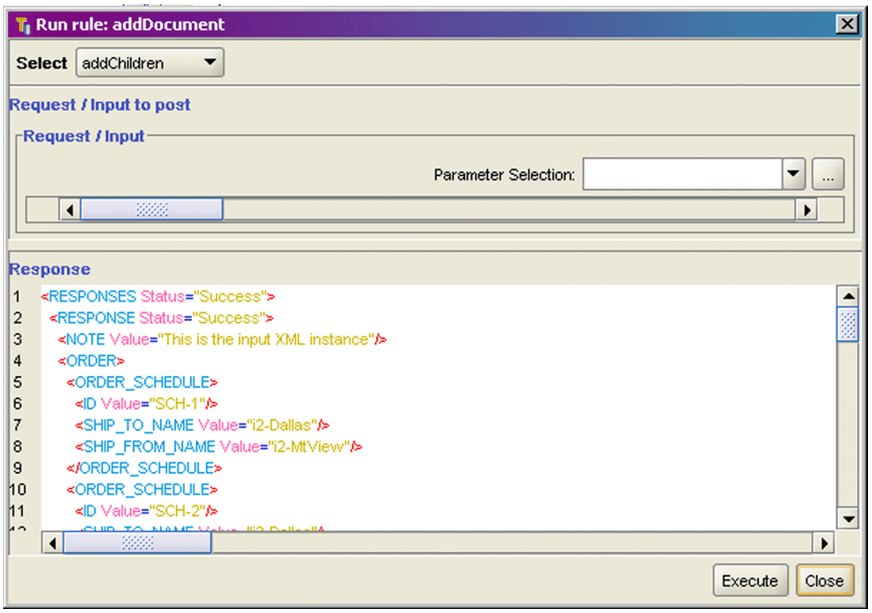
To execute a rule:

1. Start Studio Locator.
2. Launch Studio and open the ruleLangExamples.owb project file available at the following location.

   `<STUDIO_HOME>\services\studio\samples\ruleLangExamples\`
3. Start Local Server within Studio.
4. Double click on a rule file in the Rules folder in the left navigation pane. The operation/rule in the respective rule file are displayed in the bottom section of the left navigation pane.
5. Double click on the operation/rule to view its definition in the right pane; examine the syntax of the operation/rule carefully to understand its usage.
6. Right click on the operation/rule in the bottom left navigation pane and click Run.

   The Run rule dialog is displayed.

   Figure 103: Run Rule




7. Click **Execute** to carry out the operation/rule. The response of the operation/rule is displayed in the **Response** section in the **Run rule** dialog.
8. Click **Close** to close the **Run rule** dialog.

# APPENDIX I X-Service APIs and WSDLs

### What's In This Appendix

This appendix provides information on API Documents.

Topics include:


- Introduction
- API_DOC Interface Description Language
- API_DOC Language Extensions
- Generating Xservice WSDL
- Validating Xservice APIs at Runtime
- Adding Custom Extensions to API_DOC


## Introduction

The widespread adoption of XML as a flexible and non-proprietary way of describing data has simplified the integration of enterprise applications. The Web Services Description Language (WSDL) builds on this success by using XML to convey not only data but also actions to be carried out on the data.

WSDL is an XML-based language for describing web services and how to access them. It specifies the location of the service and the operations (or methods) the service exposes. A web service is a programmatic interface exposed through WSDL. This interface is described using the XML Schema Definition (XSD) language.

Given that Xcore already passes messages using XML, it is easy to see why Xcore and WSDL are a natural fit. Any Xservice API can be exposed as a web service simply by defining its interface in the WSDL description for that Xservice. The only thing that's missing is the interface description for those public APIs.

However, using the XSD language to describe Xservice APIs presents a number of problems. For the more complex Xservice APIs, one cannot form an XSD that correctly specifies the interface. This is mainly due to the inability of XSDs to conditionally include a tag (or set of tags) based on the value of an attribute. At the same time, XSDs impose an ordering on the XML which is not required by the Xservice API and which conflicts with the way APIs are invoked by legacy code. Add to this the fact that XSDs can be difficult to read and understand, and it becomes apparent that Xcore programmers need an alternate interface description language to define the Xservice APIs.

# API_DOC Interface Description Language

The API_DOC Interface Description Language (IDL) is an XML-based language that is used to document Xservice APIs. It supplies a description of the interface in an XML format that is natural for Xcore programmers to read and maintain. It also serves as an example of the input and outputs of an API and can be used to generate documentation. It contains hints which are used to generate a best-fit XSD and Validation Framework Spec file. It can be used to generate test cases for the API signature. And it provides a single point of maintenance so that all of these generated documents can remain synchronized.

Before describing the form and content of the API_DOC IDL, it might be useful to show an example of its use:

```
<DEFINE_METHOD Name="createTPA" Access="Public">
  <API_DOC>
    <INPUT>
      <REQUEST Name="createTPA" Any="true" Comment="This API
      creates TPA">
        <TPA>
          <TPA_ID Value="string" Optional="true"/>
          <AUTO_HANDLING Value="string" Optional="true"/>
          <CMI_ER_TYPE Value=
          "SHIP_TO_FIRM|UNIQUE_IDENTIFIER" Optional="true"/>
          <CUST_NAME Value="string" Optional="true"/>
          <CUST_ORG_ID Value="string"/>
          <DAILY_PLANNING_WEEKS Value="5[Type=int]"
          Optional="true"/>
          <DESCRIPTION Value="string" Optional="true"/>
          <END_DATE Value="01/01/2004
          10:23:42[Type=datetime]" Optional="true"/>
          <FCST_DOC_META_ID Value="string" Optional="true"/>
          <FIRM_IDENTIFIER_TYPE Value="string"
          Optional="true"/>
          <START_DATE Value="01/01/2004
          10:23:42[Type=datetime]" Optional="true"/>
          <STATUS Value="string" Optional="true"/>
          <SUPP_NAME Value="string" Optional="true"/>
          <SUPP_ORG_ID Value="string"/>
          <TPA_NAME Value="string"/>
          <TPA_TYPE Value="VMI|CMI_ER"/>
        </TPA>
```

```
    </REQUEST>
  </INPUT>
  <OUTPUT>
    <ON_SUCCESS>
      <RESPONSE Status="Success[Type=Constant,
      Optional=true]"Any="true" OrderIndicator="true"
      Comment="Success response of creates TPA">
        <TPA_ID Value="1"/>
        <ANY/>
      </RESPONSE>
    </ON_SUCCESS>
    <ON_EXCEPTION>
      <RESPONSE XsdDef="createTPAError"
      Comment="Validation error response of create TPA"/>
    </ON_EXCEPTION>
  </OUTPUT>
  </API_DOC>
  <RULE>
    <ACTION>
       [rule logic]
    </ACTION>
  </RULE>
</DEFINE_METHOD>
```

The example shown above defines the interface for the createTPA API. The API_DOC has an Input section to define the API Request and an Output section to define the API Response. The Output section has an On_Success section to define the Response of a successful API invocation and an optional On_Exception section to define the Response of an API invocation that returns with Status equal to ‘Error’.

## Specifying the Data Type of an Attribute

Each attribute specified in the interface definition can include a data type declaration. If none is specified, the value of the attribute is defaulted to the string data type. The list of recognized data types are: string, LongString, int, long, float, double, date, datetime, clob, boolean, xml and timestamp. For details on the acceptable formats for these data types, refer to the Xcore Programmer’s Guide.

The following example illustrates an attribute declaration with the default string data type:

```
<START_DATE Value="Today"/>
```

The following example illustrates an attribute declaration with a data type and an example value for the attribute:

```
<START_DATE Value="01/01/2004 10:23:42[Type=datetime]"/>
```

In addition to defined data types, an attribute can be declared to have a constant value or to be an enumerated type.

The following example illustrates an attribute declaration for an enumerated type:

```
<TPA_TYPE Value="VMI|CMI_ER"/>
```

The following example illustrates an attribute declaration for a constant value:

```
<RESPONSE Status="Success[Type=Constant]" >
```

## Specifying that an Attribute is Optional

By default, an attribute declared in the interface definition is mandatory. You can specify that an attribute is optional by providing the Optional modifier in the data type declaration as shown below.

```
<CREATION_DATE Value="01/01/2004 10:23:42[Type=datetime,
Optional=true]"/>
```

## Specifying that Tag is Optional

By default, a tag declared in the interface definition is mandatory. You can specify that a tag is optional by providing the Optional attribute in the tag declaration. The following example illustrates the use of the optional attribute for a tag:

```
<TPA_ID Value="string" Optional="true"/>
```

## Allowing Extra Attributes on a Tag

For XSDs, a tag cannot have attributes other than those that are declared in the interface definition. To allow additional tags that are not validated by the XSD, provide the Any attribute in the tag declaration as shown below.

```
<RESPONSE Status="Success[Type=Constant, Optional=true]" Any="true"
/>
```

**Note:** Note that by default the Validation Framework and Xcore ignore extra attributes on a tag. As such, this directive only applies to XSD generation.

## Specifying the Order of Child Tags

For XSDs, the default compositor for grouping tags is xsd:all. This compositor specifies that the elements are unordered. However, there are restrictions on the cases where this compositor can be used in an XSD. In the cases where unordered grouping conflicts with other XSD hints (see the ANY and CHOICE_OF tags and the Repeatable attribute below), the child tags must be declared as ordered. This is accomplished by specifying the OrderIndicator attribute on the parent tag declaration as show below.

```
<RESPONSE Status="Success[Type=Constant, Optional=true]"
OrderIndicator="true">
```

```
<TPA_ID Value="1"/>

<TPA_NAME Value="string"/>

</RESPONSE>
```

**Note:** By default the Validation Framework and Xcore do not require tags to be in any order. As such, this directive only applies to XSD generation.

## Allowing Extra Child Tags

For XSDs, only tags that are declared in the interface definition are permitted. To allow additional tags that are not validated by the XSD, insert the ANY tag as shown below.

```
<RESPONSE Status="Success[Type=Constant, Optional=true]"
OrderIndicator="true">

<TPA_ID Value="1"/>

<ANY/>

</RESPONSE>
```

**Note:** By default the Validation Framework and Xcore ignore extra tags. As such, this directive only applies to XSD generation. Also note that if used for XSD generation, the ANY tag requires that the parent tag (in this case the RESPONSE tag) has the OrderIndicator hint set to true.

## Allowing Repetitions of a Tag

By default, it is assumed that a tag occurs once and only once. It is possible to specify that a tag can occur many times by setting the Repeatable hint to true in the tag declaration (as shown below).

```
<TPAS OrderIndicator="true" TotalRowCount="1[Type=int,
Optional=true]" >

<TPA Repeatable="true" Optional="true" Any="true">

<TPA_ID Value="string"/>

<AUTO_HANDLING Value="string" Optional="true"/>

<CMI_ER_TYPE Value="string" Optional="true"/>

<CREATION_DATE Value="01/01/2004 10:23:42[Type=datetime]"
Optional="true"/>

<CUST_NAME Value="string" Optional="true"/>

<CUST_ORG_ID Value="string"/>

<TPA_NAME Value="string"/>

<TPA_TYPE Value="string"/>

</TPA>
```

```
</TPAS>
```

**Note:** If used for XSD generation, the Repeatable hint requires that the parent tag (in this case the TPA tag) has the OrderIndicator hint set to true.

## Specifying a Choice of Tags

It is possible to specify that one of several groups of tags is acceptable as input. This can be accomplished by using the `CHOICE_OF` and `CHOOSE` tags as shown below.

```
<FILTER Name="TPA_ID| TPA_TYPE" Repeatable="true"
OrderIndicator="true">

<CHOICE_OF>

<CHOOSE>

<TPA_ID Value="string" MatchBy="STARTS_WITH[Type=stringCompare]"
Repeatable="true"                                    Optional="true"
Comment="element name() = FILTER/@Name"/>

<OR OrderIndicator="true" Repeatable="true" Optional="true">

  <TPA_ID Value="string"

  MatchBy="STARTS_WITH[Type=stringCompare]"

  Repeatable="true" Optional="true" Comment="element

  name() = FILTER/@Name"/>

</OR>

</CHOOSE>

<CHOOSE>

<TPA_TYPE Value="string" MatchBy="STARTS_WITH[Type=stringCompare]"
Repeatable="true"
Optional="true" Comment="element name() = FILTER/@Name"/>

<OR OrderIndicator="true" Repeatable="true" Optional="true">

  <TPA_TYPE Value="string"

  MatchBy="STARTS_WITH[Type=stringCompare]"

  Repeatable="true" Optional="true" Comment="element

  name() = FILTER/@Name"/>

</OR>

</CHOOSE>

</CHOICE_OF>

</FILTER>
```

**Note:** If used for XSD generation, the `CHOICE_OF` / `CHOOSE` tags require that the parent tag (in this case the FILTER tag) has the OrderIndicator hint set to true.

# API_DOC Language Extensions

The API_DOC IDL can be extended using the standard Xcore extension mechanism. Thus, new constructs can be added to the language simply by registering handlers for them. Details on adding custom extensions for the API_DOC IDL can be found in "Appendix E Upload Services".

There are several language extensions that come packaged with API_DOC. Details on these are given in the following sections.

## Referencing Declarations in an Xdocument

It is often the case that the tags specified in an API map directly from tags defined in an Xdocument. Instead of declaring these tags separately, it is best to simply reference the existing definition. The REFER_DOC directive provides the means to accomplish this, as shown below.

```
<TPA>
   <REFER_DOC Name="TPA" [ServiceName="<service-name>"]>
   <MANDATORY_PROPS>
      <TPA_NAME/>
      <TPA_TYPE/>
      <CUST_ORG_ID/>
      <SUPP_ORG_ID/>
   </MANDATORY_PROPS>
   <EXCLUDE_PROPS>
      <CREATION_DATE/>
      <LAST_MOD_DATE/>
   </EXCLUDE_PROPS>
   <RENAME_PROPS>
      <TPA_NAME Alias="NAME"/>
      <SUPP_NAME Alias="SUPPLIER_NAME"/>
   </RENAME_PROPS >
   </REFER_DOC>
</TPA>
```

The REFER_DOC directive takes two attributes. The Name attribute must be set to the name of the Xdocument being referenced. The ServiceName attribute is optional, and specifies the name of the Xservice where the Xdocument is defined.

As shown, all tags defined in the Xdocument named TPA are included as optional tags by the REFER_DOC directive. There are two optional child tags that can be used to modify this default behavior. The MANDATORY_PROPS tag is used to specify the names of all the Xdocument tags which are required tags in the interface definition. The EXCLUDE_PROPS tag is used to specify the names of all the Xdocument tags which are to be excluded from the interface definition.

The REFER_DOC directive can contain one more optional child tag. There are cases where a referenced Xdocument property is called by another name in the interface definition. The RENAME_PROPS tag contains a list of Xdocument properties and their 'aliases' in the interface definition. When a property is copied, it is renamed using the defined alias.

During WSDL generation the REFER_DOC short cut will expand to the following form:

```
<TPA>
  <TPA_ID Value="string" Optional="true"/>
  <AUTO_HANDLING Value="string" Optional="true"/>
  <CMI_ER_TYPE Value="string" Optional="true"/>
  <CUST_NAME Value="string" Optional="true"/>
  <CUST_ORG_ID Value="string"/>
  <DAILY_PLANNING_WEEKS Value="5[Type=int]" Optional="true"/>
  <DESCRIPTION Value="string" Optional="true"/>
  <END_DATE Value="01/01/2004 10:23:42[Type=datetime]"
  Optional="true"/>
  <FCST_DOC_META_ID Value="string" Optional="true"/>
  <FIRM_IDENTIFIER_TYPE Value="string" Optional="true"/>
  <START_DATE Value="01/01/2004 10:23:42[Type=datetime]"
  Optional="true"/>
  <STATUS Value="string" Optional="true"/>
  <SUPP_NAME Value="string" Optional="true"/>
  <SUPP_ORG_ID Value="string"/>
  <TPA_NAME Value="string"/>
  <TPA_TYPE Value="string"/>
</TPA>
```

An alternate form of the REFER_DOC directive is shown below. In this case, only the tags specified inside INCLUDE_PROPS are copied to the interface definition. These tags are included as mandatory by default; they are included as optional if the Optional attribute is given.

```
<TPA>
  <REFER_DOC Name="TPA" [ServiceName="<service-name>"]>
  <INCLUDE_PROPS>
  <TPA_NAME/>
```

```
  <TPA_TYPE Repeatable="true"/>
  <CUST_ORG_ID/>
  <SUPP_ORG_ID Optional="true"/>
  </INCLUDE_PROPS>
  <RENAME_PROPS>
  <TPA_NAME Alias="NAME"/>
  </RENAME _PROPS>
  </REFER_DOC>
</TPA>
```

## Referencing a Search Spec

Another common case is that an API is defined by a search spec. In this case, the search spec itself defines the tags and their structure. Instead of expanding the search spec definition by hand, it is best to simply reference the existing definition. The REFER_SEARCH_FILTER directive provides the means to accomplish this, as shown below.

```
<REQUEST Name="getTPA" OrderIndicator="true"
MaxRows="10[Type=int,Optional=true]"
ReturnRowCount="yes[Type=Constant, Optional=true]"
StartAtRow="0[Type=int, Optional=true]">
  <REFER_SEARCH_FILTERS Name="PROGRAM_SEARCH"/>
</REQUEST>
```

The REFER_SEARCH_FILTER directive takes a Name attribute, which must be set to the name of the search spec being referenced.

Assume that the Xservice contains the following search spec definition:

```
<search-defs>
…
  <search-def Name=”PROGRAM_SEARCH" Document="TPA"/>
     <filter-def Name=”TPA_ID"/>
     <filter-def Name=”TPA_NAME"/>
     <filter-def Name=”TPA_TYPE"/>
     <filter-def Name=”DESCRIPTION"/>
     <filter-def Name=”START_DATE"/>
     <filter-def Name=”CUST_ORG_ID"/>
     <filter-def Name=”SUPP_ORG_ID"/>
     <filter-def Name=”STATUS"/>
     <filter-def Name=”END_DATE"/>
     <filter-def Name=”CUST_NAME" MapTo="
     CUSTOMER_EM_ORGANIZATION_LINK/NAME "/>
```

```
      <filter-def Name=”SUPP_NAME" MapTo="
      SELLER_EM_ORGANIZATION_LINK/NAME "/>
   </search-def>
   …
</search-defs>
```

During WSDL generation, the example above will be expanded to the following:

```
<REQUEST Name="getTPA" OrderIndicator="true"
MaxRows="10[Type=int,Optional=true]"
ReturnRowCount="yes[Type=Constant, Optional=true]"
StartAtRow="0[Type=int, Optional=true]"
Any="true">
<FILTER Repeatable="true" OrderIndicator="true"
Name="TPA_ID|TPA_NAME|TPA_TYPE ">
<CHOICE_OF>
<CHOOSE>
  <TPA_ID Value="string"
  MatchBy="STARTS_WITH[Type=stringCompare]"
  Repeatable="true" Optional="true" Comment="element
  name() = FILTER/@Name"/>
  <OR OrderIndicator="true" Repeatable="true"
  Optional="true">
    <TPA_ID Value="string"
    MatchBy="STARTS_WITH[Type=stringCompare]"
    Repeatable="true" Optional="true" Comment="element
    name() = FILTER/@Name"/>
  </OR>
</CHOOSE>
<CHOOSE>
  <TPA_NAME Value="string"
  MatchBy="STARTS_WITH[Type=stringCompare]"
  Repeatable="true" Optional="true" Comment="element
  name() = FILTER/@Name"/>
  <OR OrderIndicator="true" Repeatable="true"
  Optional="true">
    <TPA_NAME Value="string"
```

```
    MatchBy="STARTS_WITH[Type=stringCompare]"
    Repeatable="true" Optional="true" Comment="element
    name() = FILTER/@Name"/>
  </OR>
</CHOOSE>
<CHOOSE>
  <TPA_TYPE Value="string"
  MatchBy="STARTS_WITH[Type=stringCompare]"
  Repeatable="true" Optional="true" Comment="element
  name() = FILTER/@Name"/>
  <OR OrderIndicator="true" Repeatable="true"
  Optional="true">
    <TPA_TYPE Value="string"
    MatchBy="STARTS_WITH[Type=stringCompare]"
    Repeatable="true" Optional="true" Comment="element
    name() = FILTER/@Name"/>
  </OR>
</CHOOSE>
…
</CHOICE_OF>
</FILTER>
</REQUEST>
```

## Referencing a Validation Spec

Yet another common case is that the error responses for an API are defined as part of a Validation Framework Spec (VFS) file. In this case, the tags and their structure are defined by the output of the Validation Framework. Instead of expanding the error responses by hand, it is best to simply reference the existing definition. The REFER_VALIDATION_ERRORS directive provides the means to accomplish this, as shown below.

```
<ON_EXCEPTION>
   <REFER_VALIDATION_ERRORS Name="createTPA"/>
</ON_EXCEPTION>
```

The REFER_VALIDATION_ERRORS directive takes a Name attribute, which must be set to the name of the validation spec being referenced. This typically matches the API name.

Assume that the Xservice contains the following in the validation spec for the createTPA API:

```
<user_validations>
<user_validation Name="createTPA" ServiceName="CPM"/>
<error_codes>
<error_code Name="TPA_MISSING" Severity="ERROR"/>
<error_code Name="REQUIRED_FIELD_MISSING" Severity="ERROR"/>
<error_code Name="DUPLICATE_TPA_NAME" Severity="ERROR"/>
<error_code Name="INVALID_TPA_TYPE" Severity="ERROR"/>
<error_code Name="INVALID_CUST_ORG_ID" Severity="ERROR"/>
<error_code Name="INVALID_SUPP_ORG_ID" Severity="ERROR"/>
<error_code Name="INVALID_STATUS" Severity="ERROR"/>
<error_code Name="INVALID_VALUE" Severity="ERROR"/>
<error_code Name="INVALID_DAILY_PLANNING_WEEKS" Severity="ERROR"/>
</error_codes>
<tags>
<tag Name="TPA">
  <fields>
    <field Name="TPA_NAME" Type="string">
      <validations>
        <validation_group>
          <mandatory_validation
          Error="REQUIRED_FIELD_MISSING"/>
          <validation Expression="getFirstDoc('//CPM/
          TPA','TPA_NAME',$thisField) = null"
          Error="DUPLICATE_TPA_NAME"/>
        </validation_group>
      </validations>
    </field>
    <field Name="TPA_TYPE" Type="string">
      <validations>
        <validation_group>
          <mandatory_validation
```

```
          Error="REQUIRED_FIELD_MISSING"/>
          <validation Expression =
          "isValidValue($thisField,'VMI','CMI-ER')"
          Error="INVALID_TPA_TYPE"/>
        </validation_group>
      </validations>
    </field>
  </fields>
</tag>
…
</tags>
…
</user_validation>
</user_validations>
```

During WSDL generation, the example above will be expanded to the following:

```
<ON_EXCEPTION>
<RESPONSE>
<_RESULT Value="ERROR[Type=Constant]"
Details="true[Type=Constant,Optional=true]"
OrderIndicator="true">
<_ERROR Value="TPA_MISSING | REQUIRED_FIELD_MISSING
|INVALID_DAILY_PLANNING_WEEKS | INVALID_STATUS |INVALID_SUPP_ORG_ID
| INVALID_TPA_TYPE |INVALID_CUST_ORG_ID |INVALID_VALUE
|DUPLICATE_TPA_NAME" Optional="true" Repeatable="true"
OrderIndicator="true" Severity="ERROR[Type=Constant]">
  <ANY/>
</_ERROR>
</_RESULT>
<REQUEST Name="createTPA" Any="true" Comment="This API creates TPA">
<TPA>
<TPA_ID Value="string" Optional="true"/>
<TPA_NAME Value="string">
<_ERRORS Value="ERROR[Type=Constant]" Optional="true"
OrderIndicator="true">
```

```
<_ERROR Value="REQUIRED_FIELD_MISSING|DUPLICATE_TPA_NAME"
Optional="true"
Repeatable="true" OrderIndicator="true"
Severity="ERROR[Type=Constant]">

<ANY/>

</_ERROR>

</_ERRORS>

</TPA_NAME>

<TPA_TYPE Value="string">

<_ERRORS Value="ERROR[Type=Constant]" Optional="true"
OrderIndicator="true">

<_ERROR Value="REQUIRED_FIELD_MISSING|INVALID_TPA_TYPE"
Optional="true"
Repeatable="true" OrderIndicator="true"
Severity="ERROR[Type=Constant]">

<ANY/>

</_ERROR>

</_ERRORS>

</TPA_TYPE>

…

<_ERRORS Value="ERROR|SEVERE_ERROR" Optional="true"/>

</TPA>

<_ERRORS Value="ERROR|SEVERE_ERROR" Optional="true"/>

</REQUEST>

</RESPONSE>

</ON_EXCEPTION>
```

## Advanced Usage

- You can generate validation error response from multiple validation specs. This is very useful when you have multiple validation specs exercised on the API.
- By default REFER_VALDATION_ERRORS uses the INPUT of the current API_DOC to generate the error documentation. Optionally you can use documentation in other API_DOCS as input (VAL_INPUT) to this tag.
- By default REFER_VALIDATION_ERRORS generates the error response with RESPONSE as the root tag. Optionally you can select a portion of the output (VAL_OUTPUT) generated by this tag and use it in the current API_DOC.

*Example 1:*

```
<ON_EXCEPTION>
  <RESPONSE Status="Success[Type=Constant]">
     <_RESULT Value="ERROR|SEVERE_ERROR"/>
     <ORDER ValidationResult="ERROR|SEVERE_ERROR">
        <REFER_VALIDATION_ERRORS
        Name="orderCreateValidation|poCreateValidation|
        poItemValidation">
          <VAL_OUTPUT Path="REQUEST/ORDER"/>
          <VAL_INPUT Name="createPo" Path="INPUT"/>
        </REFER_VALIDATION_ERRORS>
     </ORDER>
  </RESPONSE>
</ON_EXCEPTION>
```

In the above example:

- The validation error response is being generated from 3 validation specs (orderCreateValidation, poCreateValidation and poItemValidation).
- The input to the validation errors is the INPUT of the api =createPo.
- This example selects only a portion of the response generated by REFER_VALIDATION_ERRORS. It picks up all the child tags of `REQUEST/ORDER` under the root tag `RESPONSE`.

## Referencing Other API_DOCs

Sometimes the tags passed to an API are used to internally invoke another API. In this case, the definition of these tags and the hints associated with them are actually already defined. The REFER_API_DOC directive can be used to reference the definition of another API and inline the definition.

```
<REQUEST Name="attachBufferSLAsWithMultipleShipTos" Any="true"
Comment="Given a program and item, create multiple buffers with
settings">
  <SOURCE Value="string"/>
  <BUFFER_SLA>
     <TPA_ID Value="string"/>
     <ITEM_ID Value="string"/>
     <SHIP_TO_IDS OrderIndicator="true">
     <SHIP_TO_ID Value="string" Repeatable="true"/>
     </SHIP_TO_IDS>
     <CATEGORY_ID Value="string" Optional="true"/>
     <REFER_API_DOC Name="createCmiErSettings" Path="INPUT/REQUEST"
     [ ServiceName="CPM" ] />
```

```
      …
   </BUFFER_SLA>
</ REQUEST >
```

The example above shows how to inline the input request of the createCmiErSettings API, pulling in everything under (but not including) the REQUEST tag.

During WSDL generation, the example above will be expanded to the following:

```
<REQUEST Name="attachBufferSLAsWithMultipleShipTos" Any="true"
Comment="Given a program and item, create multiple buffers with
settings">
<SOURCE Value="string"/>
   <BUFFER_SLA>
      <TPA_ID Value="string"/>
      <ITEM_ID Value="string"/>
      <SHIP_TO_IDS OrderIndicator="true">
      <SHIP_TO_ID Value="string" Repeatable="true"/>
      </SHIP_TO_IDS>
      <CATEGORY_ID Value="string" Optional="true"/>
      <CMI_ER_SETTINGS>
      <ID Value="string" Optional="true"/>
      <FIRM_WINDOW Value="5[Type=int]"/>
      <FROZEN_WINDOW Value="5[Type=int]"/>
      <IS_CATEGORY Value="string" Optional="true"/>
      <NAME Value="string" Optional="true"/>
      <CMI_VIOLATIONS OrderIndicator="true">
        <CMI_VIOLATION Repeatable="true">
          <ID Value="string" Optional="true"/>
          <VIOLATION_NAME Value="string"/>
          <RESOLUTION Value="string"/>
        </CMI_VIOLATION>
      </CMI_VIOLATIONS>
      </CMI_ER_SETTINGS>
      …
   </BUFFER_SLA>
</ REQUEST >
```

# Generating Xservice WSDL

Xcore is now packaged with a utility to generate WSDL descriptions for the APIs exposed by an Xservice. There are two prerequisites to using the utility:

1. Each Xservice API must be assigned an access privilege. There are 3 levels of access possible:
   - Public: Can be invoked by all clients.
   - Protected: Can only be invoked by the owning Xservice and other co-located Xservices. (This is the default access level.)
   - Private: Can be invoked only by the owning Xservice.
2. Any Xservice method that is to be exposed as a web service must specify its interface description using API_DOC.

The following code segment illustrates how to assign an access privilege to an API:

```
<DEFINE_METHOD Name="createTPA" Access="Public">
   <API_DOC>
      …
   </API_DOC>
   <RULE>
      …
   </RULE>
      …
</DEFINE_METHOD>
```

Once the Xservice APIs have been assigned access privileges and the interfaces have been defined using API_DOC, the Xservice WSDL can be generated using the MDM Studio. For more information refer to the *MDM Platform Studio User Guide*.

The SOAP address is used to define the Xservice WSDL. The output directory is the root directory for all generated files. The WSDL generator will create a subdirectory for each Xservice, and store the generated WSDL files appropriately. Also note that the output directory is cleared each time the WSDL generator is run.

The SOAP address value and output directory are configured using the X-Server configuration file.

```
<wsdlGenerate>
  <soapAddr Value="http://localhost:4444"/>
  <outputDir Value="C:\Temp\WSDL"/>
</wsdlGenerate>
...
</dom-config>
```

For each Xservice API that has its interface defined using API_DOC, the WSDL generator will create:

- An XML file containing the `expanded' form of the API_DOC,
- An XSD file based on the expanded API_DOC XML, and
- A VFS file based on the expanded API_DOC XML.

The generated Xservice WSDL files can now be used to perform validation on the APIs as explained in “Appendix D  Validation Services”, generate API documentation and/or test cases, etc.

### API Access Control on MDM Services

The API access control works only when the security is turned on that service in MDM.

To enable this, add SecurityCheck= “true” in service config.

For example: To toolkit.xml, add `SecurityCheck="true"`

```
<?xml version="1.0" encoding="UTF-8"?>
<dom-config>
<service-config Name="BCM_MASTER" TransactionCache="false"
Register="true" SecurityCheck="true">
```

# Validating Xservice APIs at Runtime

The files that are generated by the WSDL generator can be used to validate the APIs of an Xservice. This is accomplished by providing the optional ValidationType attribute on the service-config element in the Xservice config file. The value of this attribute is either XSD or VFS.

```
<SERVICE-CONFIG Name="CPM" TransactionCache="true"
ValidationType="XSD">
```

OR

```
<SERVICE-CONFIG Name="CPM" TransactionCache="true"
ValidationType="VFS">
```

When a valid validation type is given in the Xservice config file, the requests and responses for all public APIs in that Xservice will be validated using the appropriate mechanism. In addition, any other APIs which have an interface defined using API_DOC will also be validated.

Validation will fail and an exception will be issued if:

1. An API declared as public access is missing an interface definition.
2. The input request doesn’t conform to the interface definition.
3. The output response doesn’t conform to the interface definition.

In addition to enabling validation in the Xservice config, there are several other steps which must be performed to get runtime validation of the APIs to work. These steps differ based on the type of validation being performed.

For XSD validation:

1. The XSD Validator searches the classpath for <service-name>/<service-name>.xsd. Thus, the root directory of the generated WSDL files must be added to the classpath. Note that this is the same path that is specified as the output directory in the xserver.xml file for WSDL generation.
2. The XSD Validator requires Xerces version 2.6 and associated JARs. This means that the xercesImpl.jar and xml-apis.jar files from the TDSCOS VOB must be added as libraries in the classpath.
3. The XSD Validator loads XSDs for Xservice APIs at each invocation.

For VFS validation:

1. The VFS Validator uses the Validation Framework to validate the Xservice APIs. Thus, the validation spec files that are generated by the WSDL generator must be specified in the validationSpecFiles section of the Xservice config file as follows:

```
<DOM-CONFIG>
   <SERVICE-CONFIG Name="CPM" TransactionCache="true"
   ValidationType="VFS">
      …
      <VALIDATIONSPECFILES>
        <VALIDATIONSPECDIR Name="CPM" Extension="vfs">
        …
      </VALIDATIONSPECFILES>
   …
   </SERVICE-CONFIG>
</DOM-CONFIG>
```

   **Note:** The extension attribute is optional and defaults to ‘xml’. Also, the assumption is that the directory named ‘CPM’ is found in the classpath. Thus, as with XSD validation, the root directory of the generated WSDL files should be added to the clasps. Note that this is the same path that is specified as the output directory in the xserver.xml file for WSDL generation.
2. The VFS Validator loads all of the validation spec files at init. The names of the APIs in the generated files are changed to avoid conflicts with user-defined validation specs.

# Adding Custom Extensions to API_DOC

Custom extensions to the API_DOC IDL can be added using the standard Xcore extension mechanism. Simply create a handler for the custom tag using the DocGen interface. Then add the handler to the list of DocGen handlers as shown below. Refer to the existing handlers for details on how to implement a new one.

Inside com/teradata/xcore/wsdl/docgen_extension.xml:

```
<XCORE_EXTENSION>
  <DOCGEN_EXTENSION>
     <REFER_SEARCH_FILTERS
     ClassName="com.teradata.xservice.orderadmin.base.search.def.Ref
     erSearchFiltersHandler"/>
     <REFER_VALIDATION_ERRORS
     ClassName="com.teradata.xservice.validation.ReferValErrorsHandl
     er"/>
     …
  </DOCGEN_EXTENSION>
</XCORE_EXTENSION>
```

Inside xserver.xml:

```
<DOM-CONFIG>
…
<EXTENSIONFILES>
…
<EXTENSIONFILE Name="com/teradata/xcore/wsdl/docgen_extension.xml"/>
…
</EXTENSIONFILES>
…
</DOM-CONFIG>
```

# APPENDIX J Data Relationship Management APIs

### What's In This Appendix

This appendix describes the APIs of Master Data Relationship Management and Cross Reference.

Topics include:


- Manage Relationship Objects
- Manage Filter
- Manage Relationship
- Manage Cross Reference
- Hierarchy Manager Rules


## Manage Relationship Objects

The manage relationship object section describes the following APIs:

### Create

- Name: addRelationObject
- Purpose: this request is used to create objects.
- Input Parameters: Object Name, Object Description, Logical Table Name, Display Column Name, MasterDataUrl, Url Parameter, Sort Order, IsView, Logical Display Column, Logical Sort Column, PKCOl_Name_Log_Phy, Name, Column Name, Type, Is PrimaryKey.
- Example:

```
<REQUEST Name="addRelationObject">
    <objectName Value="AttributeSetObject"/>
    <objectDesc Value="AttributeSetObject"/>
    <tableName Value="AttributeSet"/>
    <displayColName Value="Coalesce( cast
(MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID as varchar(255)),'NULL') "/>
    <versionDisplayColName Value="Coalesce( cast
(VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID as varchar(255)) , 'NULL') "/>
    <masterDataUrl Value="/mdm/bcm/framework/tableeditor/
tableeditor.jsp?DIRECTORY=bcm.framework.tableeditor&amp;FILE=TableEdi
torView&amp;MODE=START&amp;TABLE_NAME=AttributeSet&amp;SERVICE=BCMMas
terService&amp;Auth_Enabled=true&amp;DO_SEARCH=Yes"/>
```

```
    <versionMasterDataUrl Value="/mdm/bcm/framework/tableeditor/
tableeditor.jsp?DIRECTORY=bcm.framework.tableeditor&amp;FILE=TableEdi
torView&amp;MODE=START&amp;TABLE_NAME=AttributeSet&amp;SERVICE=BCMVer
sionService&amp;Auth_Enabled=true&amp;DO_SEARCH=Yes"/>
    <urlParamter Value="F_attributeSetID"/>
    <sortOrder Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <VersionSortOrder Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <isView Value="false"/>
    <logicalDisplayColumn Value="attributeSetID"/>
    <logicalSortColumn Value="attributeSetID"/>
    <versionPhyTablename Value="VER_ATTRIBUTE_SET"/>
    <hoverPhyColName Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <hoverVersionPhyColName
Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <versionViewName Value=""/>
    <versionViewDisplayColName Value="Coalesce( cast
(.ATTRIBUTE_SET_ID as varchar(255)) , &apos;NULL&apos;) "/>
    <VersionViewSortOrder Value=".ATTRIBUTE_SET_ID"/>
    <hoverViewPhyColName Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <hoverViewVersionPhyColName Value=".ATTRIBUTE_SET_ID"/>
    <PKCOL_NAME_LOG_PHY>
    <PROPERTY Name="attributeSetID" ColumnName="ATTRIBUTE_SET_ID"
Type="string" PrimaryKey="yes"/>
    </PKCOL_NAME_LOG_PHY>
</REQUEST>
```

## Edit

- Name: updateRelationalObjectTable
- Purpose: this request is used to edit objects.
- Input Parameters: RoId, Object Description, Url Value, Url Parameter, Physical Sort, Logical Display Column, Physical Display column, Logical Sort Column.
- Example:

```
<REQUEST Name="updateRelationalObjectTable">
    <RoId Value="1201"/>
    <ObjectDescription Value="AttributeSetObjectMod"/>
    <Url Value="/mdm/bcm/framework/tableeditor/
tableeditor.jsp?DIRECTORY=bcm.framework.tableeditor&amp;FILE=TableEdi
torView&amp;MODE=START&amp;TABLE_NAME=AttributeSet&amp;SERVICE=BCMMas
terService&amp;Auth_Enabled=true&amp;DO_SEARCH=Yes"/>
    <UrlParam Value="F_attributeSetID"/>
    <PhysicalSort Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
    <logicalDisplay Value="attributeSetID , updatedLogicalTableName"/>
    <PhysicalDisplay Value="Coalesce( cast
(MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID as varchar(255)) ,
&apos;NULL&apos;) || &apos; , &apos; || Coalesce( cast
(MST_ATTRIBUTE_SET.UPDTD_LOG_TABLE_NAME as varchar(255)) ,
&apos;NULL&apos;)"/>
    <logicalSort Value="attributeSetID"/>
    <PhysicalDisplayHover Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID
|| MST_ATTRIBUTE_SET.UPDTD_LOG_TABLE_NAME"/>
    <VersionPhysicalDisplayHover
Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID ||
VER_ATTRIBUTE_SET.UPDTD_LOG_TABLE_NAME"/>
    <VersionPhysicalSort Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID"/>
```

```
    <VersionPhysicalDisplay Value="Coalesce( cast
(VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID as varchar(255)) ,
&amp;apos;NULL&amp;apos;) || &amp;apos; , &amp;apos; || Coalesce( cast
(VER_ATTRIBUTE_SET.UPDTD_LOG_TABLE_NAME as varchar(255)) ,
&amp;apos;NULL&amp;apos;)"/>
    <VERSION_URL/>
</REQUEST>
```

## Delete

- Name: deleteRelationalObject
- Purpose: this request is used to delete MDRM objects.
- Input Parameters: RO_ID
- Example:

```
<REQUEST Name="deleteRelationalObject">
    <RO_ID Value="1201"/>
</REQUEST>
```

## Input Parameters Description

The following table describes the input parameters used in Manage Object APIs.

| Input Parameters | Description |
|---|---|
| Object Name | Name of the object. |
| Object Description | Description of the object. |
| Logical Table Name | Logical table name of the object. |
| Display Column Name/Physical Display Column | Display column physical name. |
| MasterDataUrl | URL to be launched from hierarchy viewer. By default MDM table editor will be populated. |
| UrlParameter | Primary key logical name which will be passed as search condition while launching the above said URL. |
| Sort Order/Physical Sort | Physical column name, which will be used for sorting while selecting the data from the actual table. |
| IsView | Object data source in case of view that value has to be true. |
| Logical Display Column | Logical column name which will be used as node information in hierarchy. |
| Logical Sort Column | Logical column on which sorting of data is done. |
| <PKCOL_Name_LOG_PHY> | This tag should have detailed information of link column. |
| Name | Logical Column |
| Column Name | Physical column name |
| Type | Data type |

| Input Parameters | Description |
|---|---|
| Is PrimaryKey | Yes or No |
| RoId/RO_ID | Relational Object Id. |
| VersionSortOrder | Corresponding version column of SortOrder column. |
| VersionPhyTablename | Corresponding version column of PhyTablename column. |
| HoverPhyColName | This column entry is used for displaying name for tables. |
| HoverVersionPhyColName | Corresponding version column of HoverPhyColName. |
| VersionViewName | Corresponding version column of ViewName column. |
| VersionViewDisplayColName | Corresponding version column of ViewDisplayColName. |
| VersionViewSortOrder | Corresponding version column of ViewSortOrder. |
| HoverViewPhyColName | This column entry is used for displaying displaycolumn name for views. |
| HoverViewVersionPlyColName | Corresponding version column of HoverViewPhyColName. |

# Manage Filter

The manage filter section describes the following APIs:

## Create

- Name: addRelationObjectFilters
- Purpose: this request is used to create filters.
- Input Parameters: RoId, Filter Name, Logical Filter Condition, Filter Description, Filter Condition.
- Example:

```
<REQUEST Name="addRelationObjectFilters">
    <FILTER_DETAILS>
        <RoId Value="1201"/>
        <FilterName Value="AttributeSetFilter"/>
        <LogicalFilterCondition Value="AttributeSet.attributeSetID =
&apos;testName&apos; "/>
        <FilterDescription Value="AttributeSetFilter"/>
        <FilterCondition Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID =
&apos;testName&apos; "/>
        <FilterConditionVersion
Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID = &apos;testName&apos; "/>
    </FILTER_DETAILS>
</REQUEST>
```

## Update

- Name: updateRelationObjectFilters

- Purpose: this request is used to edit filters.
- Input Parameters: Logical Filter Condition, Filter Condition.
- Example:

```
<REQUEST Name="updateRelationObjectFilters">
    <UPDATE_TEMP_XML>
       <LogicalFilterCondition Value="AttributeSet.attributeSetID =
&apos;testName&apos;  OR AttributeSet.attributeSetID IS NOT NULL  "/>
       <FilterCondition Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID =
&apos;testName&apos;  OR MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID IS NOT
NULL  "/>
       <FilterConditionVersion
Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID = &apos;testName&apos;  OR
VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID IS NOT NULL  "/>
    </UPDATE_TEMP_XML>
    <SELECTED_FILTER_ID Value="11"/>
    <UPDATE_PROPERTIES_XML>
    <LogicalFilterCondition Value="AttributeSet.attributeSetID =
&apos;testName&apos;  OR AttributeSet.attributeSetID IS NOT NULL  "/>
    <FilterCondition Value="MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID =
&apos;testName&apos;  OR MST_ATTRIBUTE_SET.ATTRIBUTE_SET_ID IS NOT
NULL  "/>
    <FilterConditionVersion Value="VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID
= &apos;testName&apos;  OR VER_ATTRIBUTE_SET.ATTRIBUTE_SET_ID IS NOT
NULL  "/>
    </UPDATE_PROPERTIES_XML>
</REQUEST>
```

## Delete

- Name: deleteRelationObjectFilter
- Purpose: this request is used to delete filters.
- Input Parameters: SELECTED_FILTER_ID
- Example:

```
<REQUEST Name="deleteRelationObjectFilters">
              <SELECTED_FILTER_ID Value="11"/>
</REQUEST>
```

## Input Parameters Description

The following table describes the input parameters used in Manage Filter APIs.

| Input Parameters | Description |
|---|---|
| RoId | Relational Object Id. |
| Filter Name | Name of the filter. |
| Logical Filter Condition | Filter condition with logical column names. |
| Filter Description | Description of the filter. |
| Filter Condition | Filter condition with physical column names. |

| Input Parameters | Description |
|---|---|
| Selected_Filter_Id | Selected filter Id. |
| FilterConditionVersion | Filter condition for versioned columns |

# Manage Relationship

The manage relationship section describes the following APIs:

## Create

- Name: addRelationDetails
- Purpose: this request is used to create relationship between two objects.
- Input Parameters: Relation_Name, Relation_Desc, Parent_Object_Name, Child_Object_Name
- Example 1: use case of Relationship Data (MDM Rodv)

```
<REQUEST Name="addRelationDetails">
              <RELATIONS>
                <RELATION_DETAILS>
                  <RELATION_NAME Value="AttributeRelation"/>
                 <RELATION_DESC Value="AttributeRelation"/>
                 <RELATION_TYPE Value="Other"/>
                  <PARENT_OBJECT_NAME Value="AttributeSet"/>
                  <CHILD_OBJECT_NAME Value="AttributeSetDetail"/>
                  <RELATIONSHIP_OBJECT_NAME/>
                  <NO_EXISTING Value="mdmRodvTrue"/>
                </RELATION_DETAILS>
              </RELATIONS>
              <PROPERTY/>
</REQUEST>
```

**Note:** `RELATION_TYPE` tag can be either 'Other' or 'NNW' (for Network type Relations).

- Example 2: use case of Pre-defined Relationship Object

```
<REQUEST Name="addRelationDetails">
  <RELATIONS>
    <RELATION_DETAILS>
      <RELATION_NAME Value="AttributeSet_AttributeSet_CustomRodv"/>
      <RELATION_DESC Value="Custom Rodv Pre-defined Object "/>
     <RELATION_TYPE Value="Other"/>
      <PARENT_OBJECT_NAME Value="AttributeSet"/>
      <CHILD_OBJECT_NAME Value="AttributeSetDetail"/>
      <RELATIONSHIP_OBJECT_NAME Value="CustomRodvTable"/>
      <NO_EXISTING Value="customRodvTrue"/>
    </RELATION_DETAILS>
  </RELATIONS>
  <COLUMN_MAP>
    <PARENT_CHILD_TABLE_COLUMNS>
      <PARENT_TABLE_COLUMN>
        <PARENT_COL1 Value="ParentCol1"/>
        <PARENT_COL2/>
```

```
        <PARENT_COL3/>
        <PARENT_COL4/>
        <PARENT_COL5/>
        <PARENT_PHY_COL1 Value="PARENT_ROKD_VAL1"/>
      </PARENT_TABLE_COLUMN>
      <CHILD_TABLE_COLUMN>
        <CHILD_COL1 Value="ChildCol1"/>
        <CHILD_COL2 Value="ChildCol2"/>
        <CHILD_COL3/>
        <CHILD_COL4/>
        <CHILD_COL5/>
        <CHILD_PHY_COL1 Value="CHILD_ROKD_VAL1"/>
        <CHILD_PHY_COL2 Value="CHILD_ROKD_VAL2"/>
      </CHILD_TABLE_COLUMN>
    </PARENT_CHILD_TABLE_COLUMNS>
  </COLUMN_MAP>
</REQUEST>
```

**Note:** `RELATION_TYPE` tag can be either 'Other' or 'NNW' (for Network type Relations).

## Link Data

- Name: linkRodvData
- Purpose: this request is used to link data of two objects.
- Input Parameters: Parent Data Info, Child Data Info.
- Example:

```
<REQUEST Name="linkRodvData">
              <PARENT_INFO>
                <PARENT_ROW_DATA>
                  <attributeSetID Value="8"/>
                  <ENTITY_STATE Value="ACTIVE"/>
                  <SOURCE Value="BackEnd"/>
                  <description Value="ATTR_DESC_234"/>
                  <name Value="ATTR_NAME_234"/>
                  <updatedLogicalTableName Value="TABLE_1"/>
                </PARENT_ROW_DATA>
              </PARENT_INFO>
              <CHILDREN_INFO>
                <CHILD_ROW_DATA>
                  <CHILD_COUNTER Value="0"/>
                  <attributeSetID Value="8"/>
                  <updatedLogicalColumnName Value="attributeSetID8"/>
                  <ENTITY_STATE Value="ACTIVE"/>
                  <SOURCE Value="BackEnd"/>
                </CHILD_ROW_DATA>
                <CHILD_ROW_DATA>
                  <CHILD_COUNTER Value="1"/>
                  <attributeSetID Value="7"/>
                  <updatedLogicalColumnName Value="attributeSetID7"/>
                  <ENTITY_STATE Value="ACTIVE"/>
                  <SOURCE Value="BackEnd"/>
                </CHILD_ROW_DATA>
              </CHILDREN_INFO>
              <ROM_ID Value="1101"/>
</REQUEST>
```

### Delete

- Name: deleteRelationship
- Purpose: this request is used to delete relationship.
- Input Parameters: ROM_ID
- Example:

```
<REQUEST Name="deleteRelationship">
              <ROM_ID Value="1101"/>
</REQUEST>
```

### Input Parameters Description

The following table describes the input parameters used in Manage Relationship APIs.

| Input Parameters | Description |
|---|---|
| Relation_ Name | Name of the relation. |
| Relation_ Desc | Description of the relation. |
| Parent_Object_Name | Parent object name. |
| Child_Object_Name | Child object name. |
| ROM_Id | Relational object map Id. |
| Parent_Data_Info | Selected parent data. |
| Child_Data_Info | Selected child data. |

# Manage Cross Reference

The manage cross reference section describes the following APIs:

### Create

- Name: createCrossReference
- Purpose: this request is used to create cross reference.
- Input Parameters: Cross_Table_Name, Cross_Ref_Name, Cross_Ref_Desc, Golden_Table_Name, Is_CustomRODV.
- Example 1: use case of Relationship Data (MDM Rodv)

```
<REQUEST Name="createCrossReference">
              <CROSS_TABLE_NAMES>
                <crossRefTable1 Value="AttributeSetDetail"/>
              </CROSS_TABLE_NAMES>
              <GOLDEN_DETAILS>
                <CROSS_REF_NAME Value="AttributeCrossRef"/>
                <CROSS_REF_DESC Value="AttributeCrossRef"/>
                <GOLDEN_TABLE_NAME Value="AttributeSet"/>
              </GOLDEN_DETAILS>
```

```
                <ADDITIONAL_ATTRIBUTES/>
                <IS_CUSTOMRODV Value="false"/>
</REQUEST>
```

- Example 2: use case of Pre-defined Relationship Object

```
<REQUEST Name="createCrossReference">
  <CROSS_TABLE_NAMES>
    <crossRefTable1 Value="AttributeSetDetail"/>
  </CROSS_TABLE_NAMES>
  <GOLDEN_DETAILS>
    <CROSS_REF_NAME Value="CustomCrossRef"/>
    <CROSS_REF_DESC Value="CustomCrossRefDesc"/>
    <GOLDEN_TABLE_NAME Value="AttributeSet"/>
    <CUSTOM_TABLE Value="CustomRodvTable" IS_TABLE="true"/>
    <CUSTOM_TABLE_ID Value="123"/>
  </GOLDEN_DETAILS>
  <ADDITIONAL_ATTRIBUTES/>
  <GOLDEN_MAP_COLUMNS IS_TABLE="true" Name="AttributeSet">
    <MapParentLogColumn1 Value="ParentCol1"/>
  </GOLDEN_MAP_COLUMNS>
  <CROSS_REF_CHILDREN_MAP>
    <CROSS_REF_CHILD IS_TABLE="true" Name="AttributeSetDetail">
      <MapChildLogColumn2 Value="ChildCol2"/>
      <MapChildLogColumn1 Value="ChildCol1"/>
    </CROSS_REF_CHILD>
  </CROSS_REF_CHILDREN_MAP>
  <IS_CUSTOMRODV Value="true"/>
  <CUSTOM_RODV_NAME/>
</REQUEST>
```

## Delete

- Name: deleteCrossReference
- Purpose: this request is used to delete cross reference.
- Input Parameters: XRef_Id
- Example:

```
<REQUEST Name="deleteCrossReference">
                <XREF_ID Value="1102"/>
</REQUEST>
```

## Input Parameters Description

The following table describes the input parameters used in Manage Cross Reference APIs.

| Input Parameters | Description |
|---|---|
| Cross_Table_Name | Cross reference table name. |
| Cross_Ref_Name | Cross reference name. |
| Cross_Ref_Desc | Cross reference description. |
| Golden_Table_Name | Golden table name. |
| XRefId | Cross reference Id. |

| Input Parameters | Description |
|---|---|
| XRefDescription | Cross reference description |
| Parent_Data_Info | Selected parent data. |
| Child_Data_Info | Selected child data. |

# Hierarchy Manager Rules

This section provides the details of the format of request that needs to be posted for programmatically creating and deleting Dimension and Hierarchy through X - Rules.

## Create Hierarchy

### Request

#### *Ragged Hierarchy*

```
<REQUEST Name="createHierarchy" ServiceName="BCM_MASTER">
      <hierarchyName Value="TestHier"/>
      <hierarchyDescription Value="TestHier"/>
      <createdBy Value="USER_1"/>
      <viewRole Value="ALL"/>
      <editRole Value="ALL"/>
      <isPromotable Value="No"/>
  <ROM_IDS>
    <RomID Value="1201"/>
  </ROM_IDS>

</REQUEST>
```

#### *Unbalanced Hierarchy*

```
<REQUEST Name="createHierarchy" ServiceName="BCM_MASTER">
      <hierarchyName Value="TestHierUnbl"/>
      <hierarchyDescription Value="TestHierUnbl"/>
      <createdBy Value="USER_1"/>
      <viewRole Value="ALL"/>
      <editRole Value="ALL"/>
       <isPromotable Value="No"/>
  <ROM_IDS>
    <RomID Value="1201"/>
  </ROM_IDS>

</REQUEST>
```

### Response

```
<RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <HIERARCHY/>
    <MESSAGE Value="Hierarchy created successfully"/>
  </RESPONSE>
</RESPONSES>
```

#### *Duplicate Hierarchy Creation Error*

```
<RESPONSES Status="Error">
  <RESPONSE Status="Error" Description="[Teradata Database] [TeraJDBC
13.00.00.03] [Error 2803] [SQLState 23000] Secondary index uniqueness
violation in DEVUSER18.HIERARCHY. [BCM_MASTER]"/>
</RESPONSES>
```

## Delete Hierarchy

### Request

```
<REQUEST Name="deleteHierarchy" ServiceName="BCM_MASTER">
      <hierarchyID Value="123"/>
</REQUEST>
```

### Response

```
<RESPONSES Status="Success">
  <RESPONSE Status="Success"/>
</RESPONSES>
```

## Create Hierarchy Version

### Request

```
<REQUEST Name="createHierarchyVersion" ServiceName="BCM_MASTER">
          <hierarchyID Value="1"/>
          <versionNumber Value="2.5"/>
          <reason Value="To test Version Hierarchy"/>
          <sUserName Value="USR_1"/>
          <verStartDate Value="2009-09-16"/>
          <verEndDate Value="2009-09-30"/>
         <versionEditable Value="0"/>
         <isPromotable Value="Yes"/>
  <endDateFormat Value="2010-09-21"/>
</REQUEST>
```

**Note:** VersionEditable (0 for non-editable and 1 for editable).

### Response

```
<RESPONSE Status="Success">
            <MESSAGE Value="Hierarchy version created successfully"/>
</RESPONSE>
```

# APPENDIX K MDM Generic APIs

### What's In This Appendix

This appendix provides details on generic MDM APIs which can be useful to programmatically provide changes to existing data on Teradata MDM.

Topics include:


- Enterprise Model (User Security) APIs
- X-Core APIs
- MDM Meta Service APIs


# Enterprise Model (User Security) APIs

This section describes the enterprise model and MDM generic APIs.

## Add Organization

- Name: addOrganization
- Purpose: this request is used to add the organization name.
- Input Parameters: Id, Name, Full_Name, Address1, Address2, Address3, Postal_Code, City, State, Country, URL, Parent_Org_Id.
- Example:

```
<INPUT>
    <REQUEST Name="addOrganizationCore">
     <ID Value="org-1001"/>
       <NAME Value="SupplyBase111"/>
     <FULL_NAME Value="SupplyBase111, Inc."/>
     <ADDRESS1 Value="11511 Luna Rd."/>
     <ADDRESS2 Value="Two td place"/>
     <ADDRESS3 Value="Floor #2"/>
     <POSTAL_CODE Value="75432"/>
     <CITY Value="Dallas"/>
     <STATE Value="USA.TX"/>
     <COUNTRY Value="USA"/>
     <URL Value="http://www.org-name.com"/>
     <PARENT_ORG_ID Value="ORG_1"/>
     <CURRENCY_CODE Value="USD"/>
     <IS_SINGLE_CORPORATE ValidValues="true|false"
DefaultValue="false"/>
     <IS_REPAIR_VENDOR ValidValues="true|false" DefaultValue="false"/>
     <IS_VRMA_REQUIRED ValidValues="true|false" DefaultValue="false"/>
    </REQUEST>
</INPUT>
```

```
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
       <RESPONSE Status="Success">
       <ID Value="org-1001"/>
       </RESPONSE>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Modify Organization

- Name: modifyOrganization
- Purpose: this request is used to modify the organization details.
- Input Parameters: Id, Name, Full_Name, Address1, Address2, Address3, Postal_Code, City, State, Country, URL.
- Example:

```
<INPUT>
       <REQUEST Name="modifyOrganization">
              <ID Value="org-1001"/>
             [<ADDRESS_ID Value="addr-4931114665113892"/> ]
             [ <NAME Value="SupplyBase22"/> ]
             [ <FULL_NAME Value="SupplyBase22, Inc."/> ]
             [ <ADDRESS1 Value="test1test1"/> ]
             [ <ADDRESS2 Value="test1"/> ]
             [ <ADDRESS3 Value="test1"/> ]
             [ <POSTAL_CODE Value="56464764"/> ]
             [ <CITY Value="Dallas"/> ]
             [ <STATE Value="USA.MD"/> ]
             [ <COUNTRY Value="USA"/> ]
             [ <URL Value="http://www.org-name.com"/> ]
      </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
      <RESPONSES Status="Success">
        <RESPONSE Status="Success"/>
      </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Organization

- Name: getOrganization
- Purpose: this request is used to get the organization details.
- Input Parameters: Id
- Example:

```
<INPUT>
    <REQUEST Name="getOrganization">
       <ID Value="org-1"/>
       [ <INCLUDE_NAME/> ]
       [ <INCLUDE_NON_ACTIVE/> ]
       [ <VERSION Value="1"/> ]
```

```
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
          <RESPONSE Status="Success">
           <ORGANIZATION ExistingDocument="yes">
             <ID Value="org-1000"/>
             <ADDRESS_ID Value="addr-4921114664716398"/>
             <FULL_NAME Value="SupplyBase111, Inc."/>
             <NAME Value="SupplyBase111"/>
             <PARENT_ORG_ID Value="ORG_1"/>
             <URL Value="http://www.org-name.com"/>
             <ADDRESS1 Value="11511 Luna Rd."/>
             <ADDRESS2 Value="Two td place"/>
             <ADDRESS3 Value="Floor #2"/>
             <CITY Value="Dallas"/>
             <COUNTRY Value="USA"/>
             <CURR_VER Value="1"/>
             <ORG_ID Value="org-1000"/>
             <POSTAL_CODE Value="75432"/>
             <STATE Value="USA.TX"/>
             <VERSIONED_ID Value="org-1000;;null;;"/>
           </ORGANIZATION>
    </RESPONSE>
    </RESPONSES>
</ON_SUCCESS>
</OUTPUT>
```

## Manage Role Details

- Name: manageRoleDetails
- Purpose: this request is used to manage the role details.
- Input Parameters: Role_Id, Role_Name, Role_Template.Role_Id, Role_Template.Activity_Id, UserActivity.Name, Activity_Id, Access_Level.
- Example:

```
<INPUT>
      <REQUEST Name="manageRoleDetails" name="//UserSecurity/
manageRoleDetails" entireResponse="true" UserId="user_1">
     <RoleDetails>
        <ROLE_ID Value="BP" />
        <ROLE_NAME Value="ROLE.BP" />
      <Activities>
       <Activity Action="MODIFY">
         <ROLE_TEMPLATE.ROLE_ID Value="BP" Document="ROLE_TEMPLATE" />
         <ROLE_TEMPLATE.ACTIVITY_ID Value="bcmDataManagement"
Document="ROLE_TEMPLATE" />
         <UserActivity.NAME Value="bcmDataManagement"
Document="UserActivity" />
         <ACTIVITY_ID Value="bcmDataManagement" />
         <ACCESS_LEVEL Value="Read:Update:Add:Remove"/>
       </Activity>
       <Activity Action="REMOVE">
         <ROLE_TEMPLATE.ROLE_ID Value="BP" Document="ROLE_TEMPLATE" />
         <ROLE_TEMPLATE.ACTIVITY_ID Value="bcmManageMasterData"
Document="ROLE_TEMPLATE" />
```

```
        <UserActivity.NAME Value="bcmManageMasterData"
Document="UserActivity" />
        <ACTIVITY_ID Value="bcmManageMasterData" />
        <ACCESS_LEVEL Value="Read:Update:Add:Remove"/>
      </Activity>
      <Activity Action="ADD">
        <ROLE_TEMPLATE.ROLE_ID Value="BP" Document="ROLE_TEMPLATE" />
        <ROLE_TEMPLATE.ACTIVITY_ID Value="bcmManufacturingModel"
Document="ROLE_TEMPLATE" />
        <UserActivity.NAME Value="bcmManufacturingModel"
Document="UserActivity" />
        <ACTIVITY_ID Value="bcmManufacturingModel" />
        <ACCESS_LEVEL Value="Read:Update:Add:Remove"/>
      </Activity>
    </Activities>
  </RoleDetails>
  </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
     <RESPONSES Status="Success">
      <RESPONSE Status="Success">
    <ROLE_ID Value="BP"/>
    <ROLE_NAME Value="ROLE.BP"/>
      </RESPONSE>
     </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Add Role

- Name: addRole
- Purpose: this request is used to add the roles.
- Input Parameters: Name, Id.
- Example:

```
<INPUT>
     <REQUEST Name="addRole">
      <ID Value="admin"/>
        <NAME Value="System administrator"/>
     </REQUEST>
</INPUT>
<OUTPUT>
   <ON_SUCCESS>
       <RESPONSES Status="Success">
         <RESPONSE Status="Success">
           <ID Value="admin"/>
         </RESPONSE>
       </RESPONSES>
   </ON_SUCCESS>
</OUTPUT>
```

## Get Assignable Roles

- Name: getAssignableRoles
- Purpose: this request is used to get the assignable roles.

- Input Parameters: User_Grp_ID, Org_Type.
- Example:

```
<INPUT>
    <REQUEST Name="getAssignableRoles">
             <USER_GRP_ID Value="SU_UGP"/>
             <!--USER_ID Value="USR_1"/-->
           [ <ORG_TYPE ValidValues="Seller|Customer|System"
Value="Customer"/> ]
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
       <RESPONSE Status="Success">
            <USER_GRP_ASSIGNEMENT_MAP>
             <ASSIGNABLE_ROLE_NAME Value="Super User"/>
             <ASSIGNABLE_ROLE_ID Value="SU"/>
             <USER_GRP_ID Value="SU_UGP"/>
            </USER_GRP_ASSIGNEMENT_MAP>
            <USER_GRP_ASSIGNEMENT_MAP>
             <ASSIGNABLE_ROLE_NAME Value="Business Planner"/>
             <ASSIGNABLE_ROLE_ID Value="BP"/>
             <USER_GRP_ID Value="SU_UGP"/>
            </USER_GRP_ASSIGNEMENT_MAP>
            <USER_GRP_ASSIGNEMENT_MAP>
             <ASSIGNABLE_ROLE_NAME Value="Master Data Manager"/>
             <ASSIGNABLE_ROLE_ID Value="MDM"/>
             <USER_GRP_ID Value="SU_UGP"/>
            </USER_GRP_ASSIGNEMENT_MAP>
         </RESPONSE>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Roles for User

- Name: getRolesForUser
- Purpose: this request is used to get the roles for users.
- Input Parameters: User_Id
- Example:

```
<INPUT>
    <REQUEST Name="getRolesForUser">
     <USER_ID Value="USR_1"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
     <RESPONSES Status="Success">
     <RESPONSE Status="Success">
       <USER_ROLES/>
           </RESPONSE>
    </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Validate Role for User

- Name: validateRoleForUser
- Purpose: this request is used to validate the role for the user.
- Input Parameters: User_Id, Role_Id.
- Example:

```
<INPUT>
    <REQUEST Name="validateRoleForUser">
       <USER_ID Value="user_6" />
       <ROLE_ID Value="SU"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
      <RESPONSES Status="Success">
       <RESPONSE Status="Success">
        <IS_VALID Value="false"/>
       </RESPONSE>
       </RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Validate Role

- Name: validateRole
- Purpose: this request is used to validate the role.
- Input Parameters: Role_Id
- Example:

```
<INPUT>
    <REQUEST Name="validateRole">
    <ROLE_ID Value="role1"/>
     </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
    <RESPONSES Status="Success">
      <RESPONSE Status="Success">
        <IS_VALID Value="true"/>
        <ROLE_ID Value="role1"/>
        <ROLE_NAME Value="name1"/>
       </RESPONSE>
    </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get User Role

- Name: getUserRole
- Purpose: this request is used to get the user role.
- Input Parameters: Login_Name
- Example:

```
<REQUEST Name="getUserRole">
```

```
    <LOGIN_NAME Value="admin"/>
</REQUEST>

<RESPONSES Status="Success">
    <RESPONSE Status="Success">
      <ROLE_ID Value="IA"/>
    </RESPONSE>
</RESPONSES>
```

## Add User

- Name: addUser
- Purpose: this request is used to add user.
- Input Parameters: Login_Name, Org_Id, Manager_Id, Absentee_Id, Password, Locale Designation, First_Name, Last_Name, Email_Address, Phone, Fax, Address1, Address2, Address3, Postal_Code, City, State, Country, User_Grp_Id.
- Example:

```
<INPUT>
    <REQUEST Name="addUser" UserId="USR_1">
        <LOGIN_NAME Value="mdmUser1"/>
        <FIRST_NAME Value="First Name"/>
        <LAST_NAME Value="Last Name"/>
        <LOCALE Value="en_US"/>
        <ADDRESS1 Value="#9, ETERNA"/>
        <ADDRESS2 Value="Koramangala Industrial
          Layout" Optional="true"/>
        <ADDRESS3 Value="Koramangala"
          Optional="true"/>
        <ABSENTEE_ID Value="" Optional="true"/>
        <ORG_ID Value="_SU_ORG_"/>
        <CITY Value="Bangalore"/>
        <PHONE Value="9180-3025-1111" Optional="true"/>
       <FAX Value="9180-3025-1100" Optional="true"/>
        <STATE Value="IND.KRN"/>
        <POSTAL_CODE Value="560097"/>
        <MANAGER_ID Value="" Optional="true"/>
        <COUNTRY Value="IND"/>
        <DESIGNATION Value="ASST_MGR" Optional="true"/>
        <EMAIL_ADDRESS Value="admin@teradata.com"/>
        <PASSWORD Value="mdm"/>
        <FORCE_PASSWORD_CHANGE Value="true|false"/>
        <USR_GRPS_SELECTED>
            <USER_GRP_ID Value="SU_UGP"/>
            <USER_GRP_ID Value="ADMIN_UGP"/>
        </USR_GRPS_SELECTED>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
     <RESPONSES Status="Success">
     <RESPONSE Status="Success" DbgTag_="REQUEST" DbgCmd_="addUser">
     <ID Value="mdmUser1"/>
     </RESPONSE>
     </RESPONSES>
    </ON_SUCCESS>
    <ON_EXCEPTION>
```

```
     <RESPONSES Status="Success">
     <RESPONSE Status="Success" DbgTag_="REQUEST" DbgCmd_="addUser">
     <_RESULT Value="ERROR">
    <_ERROR Value="DUPLICATE_LOGIN_NAME" Severity="ERROR"/>
    </_RESULT>
    </RESPONSE>
    </RESPONSES>
</ON_EXCEPTION>
</OUTPUT>
```

## Modify User

- Name: modifyUser
- Purpose: this request is used to modify user
- Input Parameters: ID, Login_Name, Org_Id, Manager_Id, Absentee_Id, Password, Designation, First_Name, Middle_Initial, Last_Name, Email_Address, Phone, Fax, Address1, Address2, Address3, Postal_Code, City, State, Country.
- Example:

```
<INPUT>
      <REQUEST Name="modifyUser">
          <ID Value="user_1"/>
     <LOGIN_NAME Value="admin1"/>
     <ORG_ID Value="td"/>
     <MANAGER_ID Value="superuser"/>
     <ABSENTEE_ID Value="salesman-002"/>
     <PASSWORD Value="changeme" />
     <DESIGNATION Value="MGR" CodeMasterValue="true"
ValidValues="ASST_MGR|MGR|DIRECTOR|VP"/>
     <STATUS Value="ACTIVE" ValidValues="ACTIVE|INACTIVE"/>
     <FIRST_NAME Value="admin" />
     <MIDDLE_INITIAL Value="a"/>
     <LAST_NAME Value="admin" />
     <EMAIL_ADDRESS Value="admin@td.com" />
     <PHONE Value="111-111-1111" />
     <FAX Value="222-222-2222" />
     <ADDRESS1 Value="11511 Luna Rd." />
     <ADDRESS2 Value="Two td place" />
     <ADDRESS3 Value="Floor #2" />
     <POSTAL_CODE Value="75432" />
     <CITY Value="Dallas" />
     <STATE Value="TX" />
     <COUNTRY Value="USA" />
      </REQUEST>
    </INPUT>
    <OUTPUT>
      <ON_SUCCESS>
        <RESPONSES Status="Success">
          <RESPONSE Status="Success"/>
        </RESPONSES>
      </ON_SUCCESS>
    </OUTPUT>
```

## Get Users

- Name: getUsers

- Purpose: this request is used to get the users.
- Input Parameters: User_Id, Org_Id
- Example:

```
<REQUESTS>
<REQUEST Name="getUsers">
  <USER_ID Value="USR_12"/>
  <FILTER Name="ORG_ID">
    <ORG_ID Value="_SU_ORG_"/>
  </FILTER>
</REQUEST>
</REQUEST>
```

Example response:

```
   <RESPONSES Status="Success">
      <RESPONSE Status="Success"  >
         <USER_PROFILES {TotalRowCount="123"}>
               <USER_PROFILE>
                           <ID Value="usr-123"/>
                           <LOGIN_NAME Value="qbert"/>
                           <FIRST_NAME Value="Brad"/>
                           <LAST_NAME Value="Babin"/>
                           <EMAIL_ADDRESS Value="qbert@inet.com"/>
                           <ORG_NAME Value="Org #123"/>
                           <ORG_ID Value="org-123"/>
                           <MGR_FIRST_NAME Value="Fred"/>
                           <MGR_LAST_NAME Value="Quimby"/>
                           <ROLE_NAME Value="Customer Service Rep"/>
                           <ROLE_ID Value="CSR"/>
                           <STATUS Value="ACTIVE"/>
                {<ADDRESS1 Value="1234 Valley View Ln"/>
                <ADDRESS2 Value="Apt # 5678"/>
                <ADDRESS3 Value=""/>
                <ADDRESS4 Value=""/>
                <ADDRESS5 Value=""/>
                <POSTAL_CODE Value="75062"/>
                <CITY Value="Irving"/>
                <COUNTY Value="Dallas"/>
                <STATE Value="TX"/>
                <REGION Value="SE US"/>
                <COUNTRY Value="USA"/> }
                  </USER_PROFILE>
                           ...
         </USER_PROFILES>
      </RESPONSE>
</RESPONSES>
```

## Get User

- Name: getUser
- Purpose: this request is used to get user.
- Input Parameters: Id
- Example:

```
<REQUEST Name="getUser">
        <ID Value="user1"/>
</REQUEST>
```

```
ERROR_CODES: INVALID_USER_ID
```

## Validate Activity for User

- Name: validateActivityForUser
- Purpose: this request is used to validate the activity for the user.
- Input Parameters: User_Id, Activity_Id.
- Example:

```
<INPUT>
    <REQUEST Name="validateActivityForUser">
          <USER_ID Value="user6"/>
          <ACTIVITY_ID Value="view_enterprise_reports"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
       <RESPONSE Status="Success">
        <IS_VALID Value="false"/>
       </RESPONSE>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Add User Group

- Name: addUserGroup
- Purpose: this request is used to add user group
- Input Parameters: Grp_Name, Org_Id, Role_Temp_Id, Grp_Type, Description, Role_Template.
- Example:

```
<INPUT>
<REQUEST Name="addUserGroup">
 <ID Value="TEST_UGP"/>
 <GRP_NAME Value="TEST_UGP"/>
 <ORG_ID Value="_SU_ORG_"/>
 <GRP_TYPE Value="TEST"/>
 <DESC Value="Admin Group"/>
 <AUTO_ACTIVATE Value="true"/>
 <ASSIGNABLE_ROLE_TEMPLATES>
  <ROLE_TEMPLATE Value="ADMIN"/>
 </ASSIGNABLE_ROLE_TEMPLATES>
 <ASSIGN_ROLE_TEMPLATES>
  <ROLE_TEMPLATE Value="ADMIN"/>
  <ROLE_TEMPLATE Value="DFLT_R"/>
 </ASSIGN_ROLE_TEMPLATES>
</REQUEST>

</INPUT>
<OUTPUT>
      <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success">
```

```
    <USER_GROUP_ID Value="TEST_UGP"/>
  </RESPONSE>
</RESPONSES>
</ON_SUCCESS>
</OUTPUT>
```

## Create User Group

- Name: createUserGroup
- Purpose: this request is used to create user group
- Input Parameters: Grp_Name, Org_Id, Role_Temp_Id, Grp_Type, Description, Role_Template.
- Example:

```
<INPUT>
    <REQUEST Name="createUserGroup">
    <GRP_NAME Value="testing1"/>
    <ORG_ID Value="_SU_ORG_"/>
    <ROLE_TEMP_ID Value="admin"/>
    <GRP_TYPE Value="ADMIN"/>
    <DESC Value="desc"/>
    <ASSIGN_ROLE_TEMPLATES>
  <ROLE_TEMPLATE Value="SU"/>
 </ASSIGN_ROLE_TEMPLATES>
</REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <USER_GROUP_ID Value="usrGrp-11594966852619"/>
  </RESPONSE>
</RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Modify User Group

- Name: modifyUserGroup
- Purpose: this request is used to modify user group
- Input Parameters: Id, Grp_Name, Org_Id, Role_Id, Grp_Type, Description, Role_Template.
- Example:

```
<INPUT>
           <REQUEST Name="modifyUserGroup" entireResponse="true"
name="//UserSecurity/getRolesForUser" ScenarioId="0" SysSource="UI">
       <ID Value="usrGrp-11114672532787" />
         <GRP_NAME Value="newgrp-mona" />
         <ORG_ID Value="ORG_1" />
         <GRP_TYPE Value="ENTERPRISE" />
         <DESC Value="abcd" />
         <STATUS Value="ACTIVE" />
         <ASSIGN_ROLES>
          <ROLE_ID Value="MDM" />
```

```
            <ROLE_ID Value="BP" />
            <ROLE_ID Value="54362" />
           </ASSIGN_ROLES>
           <UNASSIGN_ROLES>
            <ROLE_ID Value="BOM-ROLE_ID" />
            <ROLE_ID Value="12345" />
            </UNASSIGN_ROLES>
        </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
           <RESPONSES Status="Success">
           <RESPONSE Status="Success"/>
           </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get User Groups

- Name: getUserGroups
- Purpose: this request is used to get user groups.
- Input Parameters: Grp_Name, Role_Temp_Id, Id.
- Example:

```
<INPUT>
<REQUEST Name="getUserGroups">
 <ORDER_BY>
  <ID Sort="Ascending"/>
 </ORDER_BY>
</REQUEST>
</INPUT>


<OUTPUT>
    <ON_SUCCESS>
        <RESPONSES Status="Success">
          <RESPONSE Status="Success">
       <USER_GRP_PROFILE>
           <ID Value="usrGrp-11035380349757"/>
           <DESCRIPTION Value="This is a Admin Group"/>
           <GRP_NAME Value="Admin Group"/>
           <ORG_ID Value="_SU_ORG_"/>
           <ROLE_TEMP_ID Value="SU"/>
           <STATUS Value="DORMANT"/>
           <TYPE Value="ADMIN"/>
       </USER_GRP_PROFILE>
          </RESPONSE>
         </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get User Group

- Name: getUserGroup
- Purpose: this request is used to get user group.
- Input Parameters: Id

- Example:

```
<INPUT>
    <REQUEST Name="getUserGroup">
   <ID Value="usrGrp-11594966852619"/>
    </REQUEST>
</INPUT>
<OUTPUT>
 <RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <USER_GRP_PROFILE ExistingDocument="yes">
      <ID Value="usrGrp-11594966852619"/>
      <DESCRIPTION Value="desc"/>
      <ENTITY_STATE Value="ACTIVE"/>
      <GRP_NAME Value="testing1"/>
      <IS_DEFAULT Value="false"/>
      <ORG_ID Value="_SU_ORG_"/>
      <STATUS Value="DORMANT"/>
      <TYPE Value="ADMIN"/>
      <ROLE_TEMP_ID/>
      <ASSIGNED_ROLE_TEMPLATE/>
      <ASSINABLE_USER_GRP_ROLE_TEMPLATES/>
    </USER_GRP_PROFILE>
    <ORGANIZATION ExistingDocument="yes">
      <ID Value="_SU_ORG_"/>
      <ADDRESS_ID Value="addr-111594277813142"/>
      <ENTITY_STATE Value="ACTIVE"/>
      <FULL_NAME Value="System"/>
      <NAME Value="System"/>
    </ORGANIZATION>
  </RESPONSE>
</RESPONSES>
</OUTPUT>
```

## Get User Group Details

- Name: getUserGroupDetails
- Purpose: this request is used to get the user group details.
- Input Parameters: Id
- Example:

```
<INPUT>
       <REQUEST Name="getUserGroupDetails">
             <ID Value="SU_UGP"/>
       </REQUEST>
</INPUT>
<OUTPUT>
          <ON_SUCCESS>
           <RESPONSES Status="Success">
             <USER_GRP_PROFILE>
                <ID Value="SU_UGP"/>
                <DESCRIPTION Value="SuperUser Group"/>
                <GRP_NAME Value="SU_UGP"/>
                <IS_DEFAULT Value="false"/>
                <ORG_ID Value="_SU_ORG_"/>
                <STATUS Value="ACTIVE"/>
                <TYPE Value="ADMIN"/>
                <ASSIGNED_ROLE_TEMPLATES>
```

```
                    <ROLE_TEMPLATES Value="SU" Name="ROLE.SU"/>
                    <ROLE_TEMPLATES Value="BP" Name="ROLE.BP"/>
                    <ROLE_TEMPLATES Value="MDM" Name="ROLE.MDM"/>
                  </ASSIGNED_ROLE_TEMPLATES>
                 <ORGANIZATION>
                 <ID Value="_SU_ORG_"/>
                 <ADDRESS_ID Value="addr-11067924579745"/>
                 <FULL_NAME Value="System"/>
                 <NAME Value="System"/>
                  </ORGANIZATION>
                 </USER_GRP_PROFILE>
        </RESPONSES>
       </ON_SUCCESS>
</OUTPUT>
```

## Activate User Group

- Name: activateUserGroup
- Purpose: this request is used to activate user group
- Input Parameters: User_Grp_Id
- Example:

```
<INPUT>
    <REQUEST Name="activateUserGroup">
           <USER_GROUP_ID Value="usrGrp-11114672532787"/>
           <USERS>
           <USER/>
           </USERS>
    </REQUEST>
</INPUT>
<OUTPUT>
      <ON_SUCCESS>
          <RESPONSES Status="Success">
            <RESPONSE Status="Success"/>
          </RESPONSES>
      </ON_SUCCESS>
</OUTPUT>
```

## Validate Login

- Name: validateLogin
- Purpose: this request is used to validate login.
- Input Parameters: Login_Name, Password
- Example:

```
<REQUEST Name="validateLogin">
                <LOGIN_NAME Value="admin"/>
                <PASSWORD Value="admin"/>
</REQUEST>

ERROR CODES - REQUIRED_FIELD_MISSING, INVALID_LOGIN_NAME,
INVALID_PASSWORD
```

## Change Password

- Name: changePassword
- Purpose: this request is used to change the password
- Input Parameters: Id, Old_Password, New_Password
- Example:

```
<INPUT>
    <REQUEST Name="changePassword">
     <ID Value="USR_1"/>
     <OLD_PASSWORD Value="admin"/>
     <NEW_PASSWORD Value="admin1"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
        <RESPONSES Status="Success">
          <RESPONSE Status="Success"/>
        </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Set Recovery Question

- Name: setRecoveryQuestion
- Purpose: this request is used to set the recovery question
- Input Parameters: User_Id, Pwd_Question, Pwd_Answer
- Example:

```
<REQUEST Name="setRecoveryQuestion">
        <USER_ID Value="user2"/>
        <PWD_QUESTION Value="LAST_4_DIGITS_OF_SSN"/>
        <PWD_ANSWER Value="1234"/>
</REQUEST>
```

## Get Recovery Question

- Name: getRecoveryQuestion
- Purpose: this request is used to get the recovery question
- Input Parameters: User_Id
- Example:

```
<REQUEST Name="getRecoveryQuestion">
 <USER_ID Value="USR_12"/>
</REQUEST>

<RESPONSES Status="Success">
  <RESPONSE Status="Success"/>
</RESPONSES>
```

## Add Contact

- Name: addContact
- Purpose: this request is used add contact

- Input Parameters: Id, Org_Id, First_Name, Middle_Initial, Last_Name, Email_Address, Phone, Fax, Address1, Address2, Address3, Postal_Code, City, State, Country.
- Example:

```
<INPUT>
     <REQUEST Name="addContact">
 <ID Value="con-001"/>
 <FIRST_NAME Value="test"/>
 <MIDDLE_INITIAL Value="a"/>
 <LAST_NAME Value="test"/>
 <IS_CONTACT Value="true"/>
 <EMAIL_ADDRESS Value="admin@td.com" />
 <PHONE Value="111-111-1111" />
 <FAX Value="222-222-2222" />
 <ADDRESS1 Value="11511 Luna Rd."/>
 <ADDRESS2 Value="Two td place"/>
 <ADDRESS3 Value="Floor #2"/>
 <POSTAL_CODE Value="75432"/>
 <CITY Value="Bangalore"/>
 <STATE Value="IND.KRN"/>
 <COUNTRY Value="IND"/>
 <ORG_ID Value="org01"/>
</REQUEST>
</INPUT>
<OUTPUT>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <ID Value="con-001"/>
  </RESPONSE>
</RESPONSES>
</OUTPUT>
```

## Add Contact Info

- Name: addContactInfo
- Purpose: this request is used add contact information
- Input Parameters: Id, Org_Id, Email_Address, Phone, Fax, Address1, Address2, Address3, Postal_Code, City, State, Country.
- Example:

```
<INPUT>
     <REQUEST Name="addContactInfo">
 <ID Value="con-002"/>
 <EMAIL_ADDRESS Value="admin@td.com" />
 <PHONE Value="111-111-1111" />
 <FAX Value="222-222-2222" />
 <ADDRESS1 Value="11511 Luna Rd."/>
 <ADDRESS2 Value="Two td place"/>
 <ADDRESS3 Value="Floor #2"/>
 <POSTAL_CODE Value="75432"/>
 <CITY Value="Dallas"/>
 <STATE Value="USA.TX"/>
 <COUNTRY Value="USA"/>
 <ORG_ID Value="org01"/>
</REQUEST>
```

```
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <ID Value="con-002"/>
  </RESPONSE>
</RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Modify Contact

- Name: modifyContact
- Purpose: this request is used modify contact
- Input Parameters: Id, Org_Id, First_Name, Middle_Initial, Last_Name, Email_Address, Phone, Fax, Address1, Address2, Address3, Postal_Code, City, State, Country.
- Example:

```
<INPUT>
     <REQUEST Name="modifyContact">
    <ID Value="con-002"/>
    <FIRST_NAME Value="admin"/>
    <MIDDLE_INITIAL Value="a"/>
    <LAST_NAME Value="admin" />
    <EMAIL_ADDRESS Value="admin@td.com" />
    <PHONE Value="111-111-1111" />
    <FAX Value="222-222-2222" />
    <ADDRESS1 Value="11511 Luna Rd."/>
    <ADDRESS2 Value="Two td place"/>
    <ADDRESS3 Value="Floor #2"/>
    <POSTAL_CODE Value="75432"/>
    <CITY Value="Dallas"/>
    <STATE Value="USA.TX"/>
    <COUNTRY Value="USA"/>
    <ORG_ID Value="org01"/>
</REQUEST>
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success"/>
</RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Get Contact

- Name: getContact
- Purpose: this request is used get contact
- Input Parameters: Id
- Example:

```
<INPUT>
        <REQUEST Name="getContact">
```

```
 <ID Value="con-002"/>
</REQUEST>
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <CONTACT ExistingDocument="yes">
      <ID Value="con-002"/>
      <ADDRESS_ID Value="ad-41594969812051"/>
      <EMAIL_ADDRESS Value="admin@td.com"/>
      <ENTITY_STATE Value="ACTIVE"/>
      <FAX Value="222-222-2222"/>
      <FIRST_NAME Value="admin"/>
      <IS_CONTACT Value="true"/>
      <LAST_NAME Value="admin"/>
      <MIDDLE_INITIAL Value="a"/>
      <ORG_ID Value="org01"/>
      <PHONE Value="111-111-1111"/>
      <ADDRESS1 Value="11511 Luna Rd."/>
      <ADDRESS2 Value="Two td place"/>
      <ADDRESS3 Value="Floor #2"/>
      <CITY Value="Dallas"/>
      <COUNTRY Value="USA"/>
      <ENTITY_STATE Value="ACTIVE"/>
      <POSTAL_CODE Value="75432"/>
      <STATE Value="USA.TX"/>
    </CONTACT>
  </RESPONSE>
</RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Delete Contact

- Name: deleteContact
- Purpose: this request is used delete contact
- Input Parameters: Id
- Example:

```
<INPUT>
     <REQUEST Name="deleteContact">
 <ID Value="con-002"/>
</REQUEST>
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
     <RESPONSES Status="Success">
  <RESPONSE Status="Success"/>
</RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Input Parameters Description

The following table describes the input parameters used in Enterprise Model APIs.

| Input Parameters | Description |
|---|---|
| Id | Organization Id |
| Name | Organization name |
| Full_Name | Organization full name |
| Address | Organization address. |
| State | State name |
| Country | Country name |
| City | City name |
| Postal_Code | Postal code of organization |
| URL | Organization URL |
| Parent_Org_Id | Parent Id of organization if it exists |
| Currency_Code | Currency code of the organization |
| Usr_ID | User Id |
| Role_Id | Role Id |
| Role_Name | Role Name |
| User_Grp_Id | User group Id |
| Grp_Name | User group name |
| Grp_Type | User group type |
| Desc | Description of the group |
| Org_Id | Name of the organization to which the user group belongs to |
| Login_Name | Login name |
| Password | User password |
| PWD_Question | Question asked during account creation. |
| PWD_Answer | Corresponding answer to the question. |
| First_Name | First name of the user |
| Middle_Initial | Middle name of the user |
| Last_Name | Last name of the user |
| Email_Address | Email address of the user |

# X-Core APIs

This section describes the X-core APIs.

## Add Code Master Value

- Name: addCodeMasterValue
- Purpose: this request is used to add code master value.
- Input Parameters: Id, Type_Id, Value_Id, Description, Parent_Id, Status
- Example:

```
<INPUT>
     <REQUEST Name="addCodeMasterValue">
          [ <ID Value="cmv-123"/> ]
            <TYPE_ID Value="STATE"/>
            <VALUE_ID Value="UPS"/>
            <DESCRIPTION Value="United Parcel Service"/>
          [ <PARENT_ID Value="cmv-cou36"/> ]
          [ <STATUS Value="ACTIVE"/> ]TATUS Value="ACTIVE"/> ]
      </REQUEST>
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
        <RESPONSES Status="Success">
          <RESPONSE Status="Success">
            <ID Value="cmv-123"/>
          </RESPONSE>
       </RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Modify Code Master Value

- Name: modifyCodeMasterValue
- Purpose: this request is used to modify the code master value.
- Input Parameters: Id, Type_Id, Value_Id, Description, Parent_Id
- Example:

```
<INPUT>
     <REQUEST Name="modifyCodeMasterValue">
            <ID Value="cmv-123"/>
          [ <VALUE_ID Value="UPS"/> ]
          [ <DESCRIPTION Value="United Parcel Service"/> ]
          [ <PARENT_ID Value="cmv-cou36"/> ]
    </REQUEST>
</INPUT>
<OUTPUT>
     <ON_SUCCESS>
       <RESPONSES Status="Success">
       <RESPONSE Status="Success"/>
       </RESPONSES>
     </ON_SUCCESS>
</OUTPUT>
```

## Set Code Master Value Status

- Name: setCodeMasterValueStatus
- Purpose: this request is used to set the code master value status.
- Input Parameters: Id, Status
- Example:

```
<INPUT>
    <REQUEST Name="setCodeMasterValueStatus">
         <ID Value="cmv-123"/>
         <STATUS Value="ACTIVE"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
      <RESPONSES Status="Success">
          <RESPONSE Status="Success"/>
      </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Code Master Value

- Name: getCodeMasterValue
- Purpose: this request is used to get the code master value.
- Input Parameters: Id, Type_Id, Value_Id
- Example:

```
<INPUT>
    <REQUEST Name="getCodeMasterValue">
         <ID Value="cmv-123"/>
         [ <PARENT_ID Value="cmv-cou36"/> ]
         [ <TYPE_ID Value="STATE"/> ]
         [ <VALUE_ID Value="UPS"/> ]
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
        <RESPONSE Status="Success">
          <CODE_MASTER_VALUE ExistingDocument="yes">
            <ID Value="cmv-123"/>
            <DESCRIPTION Value="United Parcel Service"/>
            <PARENT_ID Value="cmv-cou36"/>
            <STATUS Value="ACTIVE"/>
            <TYPE_ID Value="STATE"/>
            <VALUE_ID Value="UPS"/>
          </CODE_MASTER_VALUE>
        </RESPONSE>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Code Master Description

- Name: getCodeMasterDescription

- Purpose: this request is used to get the code master description.
- Input Parameters: Type_Id
- Example:

```
<INPUT>
    <REQUEST Name="getCodeMasterValue">
         <TYPE_ID Value="STATE"/>
    </REQUEST>
</INPUT>
<OUTPUT>
    <ON_SUCCESS>
       <RESPONSES Status="Success">
        <RESPONSE Status="Success">
          <CODE_MASTER_TYPE ExistingDocument="yes">
            <ID Value="cmv-123"/>
            <DESCRIPTION Value="United Parcel Service"/>
          </CODE_MASTER_TYPE>
        </RESPONSE>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Add Code Master Type

- Name: addCodeMasterType
- Purpose: this request is used to add the code master type.
- Input Parameters: Type_Id, Description, Parent_Type_Id, Type_Value_Id, Is_Editable
- Example:

```
<INPUT>
    <REQUEST Name="addCodeMasterType">
       <TYPE_ID Value="SHIP_VIA"/>
       <DESCRIPTION Value="United Parcel Service"/>
          [ <IS_EDITABLE Value="true" DefaultValue="false"
ValidValues="true|false"/> ]
          [ <PARENT_TYPE_ID Value="PARENT_SHIP_VIA" /> ]
          [ <TYPE_VALUE_ID Value="SHIP_VIA"/> ]
    </REQUEST>
</INPUT>
<OUTPUT>
       <ON_SUCCESS>
            <RESPONSES Status="Success">
              <RESPONSE Status="Success"/>
            </RESPONSES>
        </ON_SUCCESS>
</OUTPUT>
```

## Modify Code Master Type

- Name: modifyCodeMasterType
- Purpose: this request is used to modify the code master type.
- Input Parameters: Type_Id, Description, Parent_Type_Id, Type_Value_Id, Is_Editable
- Example:

```
<INPUT>
```

```
    <REQUEST Name="modifyCodeMasterType">
      <TYPE_ID Value="cmt-123"/>
      [<DESCRIPTION />]
      [<PARENT_TYPE_ID />]
      [<IS_EDITABLE />]
      [<TYPE_VALUE_ID />]
    </REQUEST>
</INPUT>
<OUTPUT>
   <ON_SUCCESS>
       <RESPONSES Status="Success">
         <RESPONSE Status="Success"/>
       </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Raise Error

- Name: Raise_Error
- Purpose: Alerts and returns the error value
- Example:

```
<IF_TEST Test="$thisReturn/AttributeSet/updatedLogicalTableName/
@Value!=$thisParam/LOGICAL_TABLE_NAME/@Value">
       <THEN>
          <RAISE_ERROR Value="Raise ERROR TEST"/>
       </THEN>
       </IF_TEST>
```

Output:

```
<RESPONSE Status="Success" DbgCmd_="RaiseErrorTest"
DbgNameAttr_="AttributeSet" DbgTag_="REQUEST">
              <_RESULT Value="ERROR">
                <_ERROR Value="Raise ERROR TEST" Severity="ERROR"/>
              </_RESULT>
            </RESPONSE>
```

## Raise Exception

- Name: Raise_Exception
- Purpose: Alerts and returns the Exception value with Xcore Trace
- Example:

```
<IF_TEST Test="$thisReturn/AttributeSet/updatedLogicalTableName/
@Value!=$thisParam/LOGICAL_TABLE_NAME/@Value">
       <THEN>
          <RAISE_EXCEPTION Value="ERROR"/>
       </THEN>
       </IF_TEST>
```

Output:

```
com.teradata.xcore.util.XcoreException: <RESPONSE Status="Success"
DbgCmd_="GET_DOCUMENT" DbgNameAttr_="AttributeSet">
   <RESPONSE Status="Success" DbgCmd_="GET_DOCUMENT"
DbgNameAttr_="AttributeSet">
   <_RESULT Value="SEVERE_ERROR">
      <_ERROR Value="ERROR" Severity="SEVERE_ERROR"/>
```

```
    </_RESULT>
    </RESPONSE>
        CommandExtensionRules.xml:149-151],
ACTION[CommandExtensionRules.xml:144-154],
RULE[CommandExtensionRules.xml:143-155], DEFINE_METHOD-
RaiseExceptionTest[CommandExtensionRules.xml:142-156], DEFINE_METHOD-
RaiseExceptionTest[CommandExtensionRules.xml:142-156], DEFINE_METHOD-
RaiseExceptionTest[CommandExtensionRules.xml:142-156]]
    at
com.teradata.xcore.tool.Persist.executeRequest(Persist.java:232)
    at
com.teradata.xcore.xserver.LocalRequestHandler.endNewRequest(LocalReq
uestHandler.java:319)
    at
com.teradata.xcore.xserver.LocalRequestHandler.request(LocalRequestHa
ndler.java:210)
    at
com.teradata.xcore.xserver.LocalRequestHandler.request(LocalRequestHa
ndler.java:175)
    at
com.teradata.xcore.net.SoapRequestHandler.request(SoapRequestHandler.
java:41)
    at
com.teradata.xcore.net.server.scheduling.Request.execute(Request.java
:89)
    at
com.teradata.xcore.net.util.RunnableTask.execute(RunnableTask.java:53
)
    at
com.teradata.xcore.net.server.scheduling.RequestProcessor$WorkerRunna
bleTask.execute(RequestProcessor.java:48)
    at
com.teradata.xcore.net.util.RunnableTask.run(RunnableTask.java:43)
    at java.lang.Thread.run(Thread.java:595)
    **********************  Xcore Trace ************************
```

## Get Document Name

- Name: Get_Document_Name
- Purpose: Gets the Logical Document name of a Table
- Example:

```
<GET_DOCUMENT_NAME TableName="MST_ATTRIBUTE_SET"
ServiceName="BCM_MASTER" AssignToVar="thisReturn"/>
```

Output:

```
<RESPONSE Status="Success" DbgCmd_="GetDocumentNameTest"
DbgTag_="REQUEST">
            <DOCUMENT_NAME Value="AttributeSet"/>
</RESPONSE>
```

## Get Links

- Name: Get_Links
- Purpose: Gets all the links defined for the specified document
- Example:

```
<GET_LINKS AssignToVar="thisReturn" ServiceName="BCM_MASTER">
       <TABLE_NAME Value="AttributeSet"/>
</GET_LINKS>
```

Output:

```
<RESPONSE Status="Success" DbgCmd_="GetLinksTest" DbgTag_="REQUEST">
              <DOCUMENTLINK Name="AttributeSetDetail">
                <MAP_PROPERTY To="attributeSetID"
From="attributeSetID"/>
              </DOCUMENTLINK>
              <DOCUMENTLINK Name="GroupClassGPMAttributeSet">
                <MAP_PROPERTY To="attributeSetID"
From="attributeSetID"/>
              </DOCUMENTLINK>
</RESPONSE>
```

## Collate Elements

- Name: Collate_Elements
- Purpose: Groups child elements of a given XML under a given key
- Example: Input:

```
       <COLLATE_ELEMENTS name="EACH_ITEM">
         <Name value="Name1"/>
         <Id value="Id2"/>
         <Description value="Description1"/>
         <Name value="Name2"/>
         <Id value="Id1"/>
         <Description value="Description2"/>
         <Name value="Name3"/>
         <Id value="Id3"/>
         <Description value="Description3"/>
     </COLLATE_ELEMENTS>
```

Output:

```
<RESPONSE>
  <ELEMENT_LIST>
    <EACH_ITEM>
      <Description value="Description1"/>
      <Name value="Name1"/>
      <Id value="Id2"/>
    </EACH_ITEM>
    <EACH_ITEM>
      <Description value="Description2"/>
      <Name value="Name2"/>
      <Id value="Id1"/>
    </EACH_ITEM>
    <EACH_ITEM>
      <Description value="Description3"/>
      <Name value="Name3"/>
      <Id value="Id3"/>
    </EACH_ITEM>
  </ELEMENT_LIST>
</RESPONSE>
```

## Ignore Equals Case

- Name: EQUALS_IGNORE_CASE

- Purpose: Compares two strings without case check and returns boolean result
- Example:

```
<IF>
    <COND>
       <OR>
        <EQUALS_IGNORE_CASE Value="{$thisParam/ENTITY_TYPE/@Value}"
ToValue="ORG" ServiceName="BCM_MASTER"/>
                <EQUALS_IGNORE_CASE Value="{$thisParam/ENTITY_TYPE/
@Value}"  ToValue="USER" ServiceName="BCM_MASTER"/>
          </OR>
       </COND>
    <THEN>
          <PRINTLN Value="VALID ENTITY TYPE"/>
    </THEN>
```

Output:

```
VALID ENTITY TYPE
```

## Get All Documents

- Name: Get_All_Doc_Names
- Purpose: Gets All documents associated with the specified service
- Example:

```
<GET_ALL_DOC_NAMES ServiceName="BCM_MASTER" AssignToVar="thisReturn"/
>
```

Output:

```
<RESPONSE Status="Success" DbgCmd_="GET_ALL_DOC_NAMES">
      <DOCUMENTS>
        <DOCUMENT Name="VersionDetails"
TableName="MST_VERSION_DETAILS" DisplayName="Version Details"/>
        <DOCUMENT Name="error_ValidGroupDimensions"
TableName="ERR_VALID_GRP_DIMENSIONS" DisplayName="Valid Group
Dimensions"/>
        <DOCUMENT Name="Process" TableName="PROCESS"
DisplayName="Process"/>
        <DOCUMENT Name="error_EntityClassification"
TableName="ERR_ENTITY_CLASSIFICATION" DisplayName="Entity
Classification"/>
        <DOCUMENT Name="error_RelationalObjectKeyType"
TableName="ERR_REL_OBJ_KEY_TYPE"
DisplayName="RelationalObjectKeyType"/>
....
...
</DOCUMENTS>
</RESPONSE>
```

## Get Document Properties

- Name: Get_Doc_Properties
- Purpose: Gets all properties/fields of the specified document
- Example:

```
 <GET_DOC_PROPERTIES ServiceName="BCM_MASTER"
TableName="AttributeSet" AssignToVar="thisReturn"/>
```

Output:

```
 RESPONSE Status="Success" DbgCmd_="GET_DOC_PROPERTIES">
      <DOCUMENT Name="AttributeSet" TableName="MST_ATTRIBUTE_SET"
DisplayName="Attribute Set">
        <PROPERTY Name="attributeSetID" ColumnName="ATTRIBUTE_SET_ID"
Type="string" PrimaryKey="yes"/>
        <PROPERTY Name="updatedLogicalTableName"
ColumnName="UPDTD_LOG_TABLE_NAME" Type="string"/>
        <PROPERTY Name="updatedPhysicalTableName"
ColumnName="UPDTD_PHY_TABLE_NAME" Type="string"/>
        <PROPERTY Name="description" ColumnName="DESCRIPTION"
Type="string"/>
        <PROPERTY Name="name" ColumnName="NAME" Type="string"/>
        <PROPERTY Name="SYS_TARGET_ID" ColumnName="SYS_TARGET_ID"
Type="int"/>
        <PROPERTY Name="SYS_AUTH_ID" ColumnName="SYS_AUTH_ID"
Type="string"/>
        <PROPERTY Name="SOURCE" ColumnName="SYS_SOURCE" Type="string"/
>
        <PROPERTY Name="CREATED_BY" ColumnName="SYS_CREATED_BY"
Type="string"/>
        <PROPERTY Name="CREATION_DATE" ColumnName="SYS_CREATION_DATE"
Type="datetime"/>
        <PROPERTY Name="ENTITY_STATE" ColumnName="SYS_ENT_STATE"
Type="string"/>
        <PROPERTY Name="LAST_MODIFIED_BY"
ColumnName="SYS_LAST_MODIFIED_BY" Type="string"/>
        <PROPERTY Name="LAST_MODIFIED_DATE"
ColumnName="SYS_LAST_MODIFIED_DATE" Type="datetime"/>
        <PROPERTY Name="SYS_TYPE" ColumnName="SYS_NC_TYPE"
Type="string"/>
        <PROPERTY Name="SYS_ERR_CODE" ColumnName="SYS_ERR_CODE"
Type="string"/>
        <PROPERTY Name="SYS_ERR_SEVERITY" ColumnName="SYS_ERR_SVRTY"
Type="string"/>
      </DOCUMENT>
    </RESPONSE>
```

## Get Folders

- Name: Get_Folders
- Purpose: Gets the folder names and the documents present inside them (folder structure of mtt docs for a model)
- Example:

  ```
  <GET_FOLDERS AssignToVar="thisReturn"/>
  ```

  Output:

  ```
  <RESPONSE Status="Success" DbgCmd_="GET_FOLDERS">
                    <ALL_FOLDERS>
                      <FOLDER Name="CRM">
                        <DOCUMENTS>
                          <DOCUMENT Name="ACCOUNT_MASTER"
  DisplayName="ACCOUNT_MASTER"/>
                          <DOCUMENT Name="HOUSEHOLD_ACTIVITY_SUMMARY"
  DisplayName="HOUSEHOLD_ACTIVITY_SUMMARY"/>
  ```

```
                        <DOCUMENT Name="INDIVIDUAL_MASTER"
DisplayName="INDIVIDUAL_MASTER"/>
                      </DOCUMENTS>
                    </FOLDER>
                    <FOLDER Name="GENERALAPPROVAL">
                      <DOCUMENTS>
                        <DOCUMENT Name="Approval_Groups"
DisplayName="Approval_Groups"/>
                        <DOCUMENT Name="Approvals"
DisplayName="Approvals"/>
                      </DOCUMENTS>
                    </FOLDER>
                    .
                    .
                    .
                  </ALL_FOLDERS>
                </RESPONSE>
```

## Get Supported Data Types

- Name: Get_Supported_Data_Types
- Purpose: Gets All supported datatypes by MDM
- Example:

  ```
  <GET_SUPPORTED_DATA_TYPES AssignToVar="thisReturn"/>
  ```

  Output:

  ```
   <RESPONSE Status="Success" DbgCmd_="GET_SUPPORTED_DATA_TYPES">
                    <datatype Name="stringlist"/>
                    <datatype Name="BinaryData"/>
                    <datatype Name="identity"/>
                    <datatype Name="time"/>
                    <datatype Name="periodDate"/>
                    <datatype Name="int"/>
                    <datatype Name="xml"/>
                    <datatype Name="date"/>
                    <datatype Name="boolean"/>
                    <datatype Name="LongString"/>
                    <datatype Name="long"/>
                    <datatype Name="decimal"/>
                    <datatype Name="double"/>
                    <datatype Name="st_geometry"/>
                    <datatype Name="periodTimestamp"/>
                    <datatype Name="char"/>
                    <datatype Name="string"/>
                    <datatype Name="datetime"/>
                    <datatype Name="float"/>
                    <datatype Name="periodTimestamp6"/>
                    <datatype Name="timestamp"/>
                    <datatype Name="encryptedString"/>
                    <datatype Name="Clob"/>
      </RESPONSE>
  ```

## Get Tables

- Name: Get_Tables

- Purpose: Gets all Tables with their properties including the presentation proprties, for the specified service
- Example:

```
<GET_TABLES ServiceName="BCM_MASTER" AssignToVar="thisReturn"/>
```

Output:

```
 <RESPONSE Status="Success" DbgCmd_="GetTablesTest"
DbgTag_="REQUEST">
              <DOCUMENTS>
                <DOCUMENT Name="AttributeSet"
TableName="MST_ATTRIBUTE_SET" DisplayName="Attribute Set">
                  <PROPERTY Name="attributeSetID" Type="string"
MaxLength="40" PrimaryKey="yes" Required="yes"
ColumnName="ATTRIBUTE_SET_ID" DisplayName="attributeSetID">
                    <PRESENTATION>
                      <DISPLAY_NAME Value="attributeSetID"/>
                      <SEQUENCE Value="0"/>
                      <EDITABLE Value="yes"/>
                      <HIDDEN Value="no"/>
                      <SORTABLE Value="yes"/>
                      <SEARCHABLE Value="yes"/>
                      <NO_WRAP Value="yes"/>
                      <I18NIZE Value="false"/>
                    </PRESENTATION>
                  </PROPERTY>
                  ....
                  .....
       <PROPERTY Name="SYS_ERR_SEVERITY" Type="string" MaxLength="30"
ColumnName="SYS_ERR_SVRTY" DisplayName="Error Severity">
                    <PRESENTATION>
                      <DISPLAY_NAME Value="Error Severity"/>
                      <EDITABLE Value="no"/>
                      <HIDDEN Value="no"/>
                      <SORTABLE Value="yes"/>
                      <SEARCHABLE Value="yes"/>
                      <NO_WRAP Value="yes"/>
                    </PRESENTATION>
                    <DATASOURCE>
                      <ENUMERATION
ValidValues="ERROR|SEVERE_ERROR|WARNING"/>
                    </DATASOURCE>
                  </PROPERTY>
                </DOCUMENT>
                <DOCUMENT ...
       </DOCUMENT>
              </DOCUMENTS>
            </RESPONSE>
```

## XML Sort

- Name: XML_Sort
- Purpose: Sorts the child elements in the XML node according to the specifed common attribute
- Example: for XML structure

```
<MDM_SERVICES>
```

```
            <SERVICE ServiceID="1" Name="USER_SECURITY"/>
            <SERVICE ServiceID="5" Name="E2E"/>
            <SERVICE ServiceID="2" Name="BCM_MASTER"/>
            <SERVICE ServiceID="8" Name="TEST"/>
            <SERVICE ServiceID="4" Name="BCM_OUTPUT_STAGING"/>
            <SERVICE ServiceID="3" Name="BCM_VERSION_STAGING"/>
            <SERVICE ServiceID="7" Name="VALIDATION"/>
            <SERVICE ServiceID="6" Name="BPE_META"/>
            <SERVICE ServiceID="9" Name="DATA_PROFILING"/>
            <SERVICE ServiceID="10" Name="MDM_HM"/>
            <SERVICE ServiceID="11" Name="TIMER_SINK"/>
            <SERVICE ServiceID="0" Name="BCM_NET_CHANGE"/>
          </MDM_SERVICES>
```

Syntax:

```
<XML_SORT SortKey="@ServiceID" SortOrder="Ascending" DataType="int"
AssignToVar="thisReturn">
       <TO_XML SelectList="$thisParam/MDM_SERVICES/*"/>
     </XML_SORT>
```

Output:

```
<RESPONSE Status="Success" DbgCmd_="XML_SORT">
         <SERVICE ServiceID="0" Name="BCM_NET_CHANGE"/>
         <SERVICE ServiceID="1" Name="USER_SECURITY"/>
         <SERVICE ServiceID="2" Name="BCM_MASTER"/>
         <SERVICE ServiceID="3" Name="BCM_VERSION_STAGING"/>
         <SERVICE ServiceID="4" Name="BCM_OUTPUT_STAGING"/>
         <SERVICE ServiceID="5" Name="E2E"/>
         <SERVICE ServiceID="6" Name="BPE_META"/>
         <SERVICE ServiceID="7" Name="VALIDATION"/>
         <SERVICE ServiceID="8" Name="TEST"/>
         <SERVICE ServiceID="9" Name="DATA_PROFILING"/>
         <SERVICE ServiceID="10" Name="MDM_HM"/>
         <SERVICE ServiceID="11" Name="TIMER_SINK"/>
       </RESPONSE>
```

## Input Parameters Description

The following table describes the input parameters used in Xcore APIs.

| Input Parameters | Description |
|---|---|
| Id | Code master Id (unique identifier) |
| Type_Id | Code master type Id |
| Value_Id | Code master value Id |
| Description | Code master description |
| Status | Status of the code master value |
| Parent_Id | Parent of the particular code master value if it exists |
| Parent_Type_Id | Parent type of a particular code master type if it exists |
| Is_Editable | Specifies if a particular column in code master type is editable |

# MDM Meta Service APIs

This section describes the MDM meta service APIs.

## Start Timer for Publication

- Name: startTimerForPublication
- Purpose: this request is used to start the timer for publication
- Input Parameters: Nil
- Example:

```
<REQUEST Name="startTimerForPublication"/>
```

## Get Publication Objects

- Name: getPublicationObjects
- Purpose: this request is used to get the publication objects
- Input Parameters: Nil
- Example:

```
<REQUEST Name="getPublicationObjects"/>
```

## Load ThesaurusDef

- Name: loadThesaurusDef
- Purpose: this request is used to load the ThesaurusDef
- Input Parameters: File value
- Example:

```
<REQUEST Name="loadThesaurusDef">
    <FILE Value="xservice/meta/xml/thesaurusDef.xml"/>
</REQUEST>
<RESPONSE Status="Success">
</RESPONSE>
```

## Load Business Rule Definition

- Name: loadBusinessRuleDefinition
- Purpose: this request is used to load the business rule definition
- Input Parameters: Nil
- Example:

```
&lt;REQUEST Name=&quot;loadBusinessRuleDefinition&quot;&gt;
  &lt;/REQUEST&gt;
```

## Delete Business Rule Definition

- Name: deleteBusinessRuleDefinition
- Purpose: this request is used to delete the business rule definition

- Input Parameters: Nil
- Example:

```
&lt;REQUEST Name=&quot;deleteBusinessRuleDefinition&quot;&gt;
  &lt;/REQUEST&gt;
```

## Update Business Rule Definition

- Name: updateBusinessRuleDefinition
- Purpose: this request is used to update the business rule definition
- Input Parameters: Nil
- Example:

```
&lt;REQUEST Name=&quot; updateBusinessRuleDefinition &quot;&gt;
  &lt;/REQUEST&gt;
```

## Refresh Business Rules

- Name: refreshBusinessRules
- Purpose: this request is used to update the business rule definition
- Input Parameters: Rule_Type
- Example:

```
<INPUT>
        <REQUEST Name="refreshBusinessRules">
          <RULE_TYPE Value="sql_validation"/>
          <RULE_TYPE Value="sql_filter"/>
        </REQUEST>
</INPUT>
<OUTPUT>
        <RESPONSES Status="Success">
          <RESPONSE Status="Success"/>
        </RESPONSES>
</OUTPUT>
```

## Create Global Temp Tables

- Name: createGlobalTempTables
- Purpose: this request is used to create global temp tables
- Input Parameters: Nil
- Example:

```
<!-- XRule to execute SP for creating Global Temp Tables. This rule is
invoked from load_correlation.bat file
      NOTE ::: This rule should only be invoked once SYS_TABLE_MAP is
populated as this SP uses
       PHY_TEMP column value in SYS_TABLE_MAP to create GTT tables.
  -->
  <REQUEST_METHOD Name="createGlobalTempTables">
    </REQUEST_METHOD>
```

## Create Auth Views

- Name: createAuthViews

- Purpose: this request is used to create auth views
- Input Parameters: Nil
- Example:

```
<REQUEST_METHOD Name="createAuthViews">
</ REQUEST_METHOD>
```

## Get Service Names

- Name: getServiceNames
- Purpose: this request is used to get the service names
- Input Parameters: Nil
- Example:

```
<REQUEST Name="getServiceNames" AssignToVar="response">
```

## Get Document Names

- Name: getDocumentNames
- Purpose: this request is used to get the document names
- Input Parameters: Nil
- Example:

```
<REQUEST Name="getDocumentNames" AssignToVar="response">
```

## Get Property Names

- Name: getPropertyNames
- Purpose: this request is used to get the property names
- Input Parameters: Document value
- Example:

```
<REQUEST ServiceName="BPE_META" Name="getPropertyNames"
AssignToVar="propertyList">
    <SERVICE Value="MDM_Service_Name"/>
    <DOCUMENT Value="Document_Name"/>
</REQUEST>
```

# APPENDIX L Non MDM APIs

### What's In This Appendix

This appendix provides details on non MDM APIs used in the customer relationship data management module of Teradata MDM.

Topics include:


- Non MDM APIs


# Non MDM APIs

This section describes all the non MDM APIs.

## Get Non MDM Document Names

- Name: getNonMDMDocumentNames
- Purpose: this request gets all the non MDM document names.
- Input Parameters: None
- Example:

```
<INPUT>
    <REQUESTS ServiceName="BPE_META">
    <REQUEST Name="getNonMDMDocumentNames"/>
    </REQUESTS>
</INPUT>

<OUTPUT>
    <ON_SUCCESS>
    <RESPONSES Status="Success">
    <RESPONSE Status="Success">
    <DOCUMENT Value="TESTPK" DbName="AK186024"/>
     <DOCUMENT Value="RSTEST" DbName="RS185144"/>
     <DOCUMENT Value="TEST" DbName="AK186024"/>
    </RESPONSE>
    </RESPONSES>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Non MDM Document Properties

- Name: getNonMDMDocumentProperties
- Purpose: this request gets the properties of all non MDM documents.
- Input Parameters: None.

- Example:

```
<INPUT>
    <REQUESTS ServiceName="BPE_META">
    <REQUEST Name="getNonMDMDocumentProperties"/>
    </REQUESTS>
</INPUT>

<OUTPUT>
    <ON_SUCCESS>
       <NON_MDM_DOCUMENTS>
       <DOCUMENT Value="AK186024.TEST" dbName="AK186024">
       <PROPERTY Name="col2" Type="string" Length="20"/>
       <PROPERTY Name="col1" Type="string" Length="10"/>
       </DOCUMENT>
       <DOCUMENT Value="AK186024.TESTPK" dbName="AK186024">
       <PROPERTY Name="col3" Type="string" Length="20"
       PrimaryKey="yes"/>
       <PROPERTY Name="col2" Type="string" Length="30"
       PrimaryKey="yes"/>
       <PROPERTY Name="col1" Type="string" Length="30"/>
       </DOCUMENT>
       <DOCUMENT Value="RS185144.RSTEST" dbName="RS185144">
       <PROPERTY Name="RScol1" Type="" Length="10"/>
       </DOCUMENT>
    </NON_MDM_DOCUMENTS>
    </ON_SUCCESS>
</OUTPUT>
```

## Get Properties of selected Non MDM Documents

- Name: getNonMDMDocumentProperties
- Purpose: this request gets the properties of non MDM documents mentioned in the requests.
- Input Parameters: Non MDM document value (name) and database name.
- Example:

```
<INPUT>
    <REQUESTS ServiceName="BPE_META">
    <REQUEST Name="getNonMDMDocumentProperties"/>
    <NonMDMDocument Value="TESTPK" DbName="AK186024"/>
    </REQUESTS>
</INPUT>

<OUTPUT>
    <ON_SUCCESS>
       <NON_MDM_DOCUMENTS>
       <DOCUMENT Value="AK186024.TESTPK" dbName="AK186024">
       <PROPERTY Name="col3" Type="string" Length="20"
       PrimaryKey="yes"/>
       <PROPERTY Name="col2" Type="string" Length="30"
       PrimaryKey="yes"/>
       <PROPERTY Name="col1" Type="string" Length="30"/>
    </DOCUMENT>
    </NON_MDM_DOCUMENTS>
    </ON_SUCCESS>
</OUTPUT>
```

## Reload Non MDM Documents

- Name: reloadNonMDMDocuments
- Purpose: this request refreshes non MDM document cache.
- Input Parameters: None.
- Example:

```
<INPUT>
    <REQUESTS ServiceName="BPE_META">
    <REQUEST Name="reloadNonMDMDocuments"/>
    </REQUESTS>
</INPUT>

<OUTPUT>
    <RESPONSES Status="Success">
    <RESPONSE Status="Success"/>
    </RESPONSES>
</OUTPUT>
```

## Is Non MDM Documents Registered

- Name: isDocumentRegistered
- Purpose: this request checks whether a non MDM table is registered or not.
- Input Parameters: Document name and database name.
- Example:

```
<INPUT>
    <REQUESTS ServiceName="BPE_META">
    <REQUEST Name="isDocumentRegistered" DbName="TEST_DB"
    Document="TEST_DOC"/>
    </REQUESTS>
</INPUT>

<OUTPUT>
<RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <DOCUMENT_REGISTERED Value="true"/>
  </RESPONSE>
</RESPONSES>
</OUTPUT>

OR

<OUTPUT>
<RESPONSES Status="Success">
  <RESPONSE Status="Success">
    <DOCUMENT_REGISTERED Value="false"/>
  </RESPONSE>
</RESPONSES>
</OUTPUT>
```

# APPENDIX M DB Based Locking Utility

### What's In This Appendix

This chapter gives you information on API locking features that have been implemented.

Topics include:


- Introduction
- Concept
- Setup


## Introduction

Xcore provides utility APIs to perform locking (using a common table in database) that can be used to avoid deadlock and wrong results due to concurrency issues.

This feature has been implemented as an extension to xcore. The setup needed to use this feature is explained in the document below.

## Concept

The locking feature has been implemented by using a table in database where all lock entries are kept. The table is named LOCK_TABLE and the structure of this table is:

Table 100: Lock_Table

| LOCK_KEY | LOCK_VALUE | LOCK_TIME |
|---|---|---|
| ITEM_ID^SHIP_TO_ID^ | V-102-1^Velocity-Tonawanda^ | 1091655519497 |
| ITEM_ID^SHIP_TO_ID^ | V-102-1^Velocity-Warren^ | 1091655519497 |
| ITEM_ID^SHIP_TO_ID^ | V-102-1^Velocity-DC1^ | 1091666519497 |
| ITEM_ID^SHIP_TO_ID^ | V-104-1^Velocity-Warren^ | 1091666519497 |

Every rule that is invoked from outside and results in a new transaction can use the locking API provided to modify a set of rows (or insert rows if not present) that correspond to the data items that it intends to work on. If two transactions try to modify the same row at same time (i.e. they want to work on same data items) then since both are trying to modify the same row

one transaction will succeed and the other transaction will be forced to wait till the first transaction is done.

# Setup

## Change xserver.xml

The xserver.xml needs to be modified and following entries need to be added.

```
<docDefnFiles>

     ...

    <docDefnFile Name="com/teradata/xcore/dblock/dblock_doc.xml"/>

    ...

</docDefnFiles>


<extensionFiles>

    ...

    <extensionFile Name="com/teradata/xcore/dblock/
dblock_extension.xml"/>

    ...

</extensionFiles>
```

## Define a lock specification file

The format of the file is:

```
<DEFINE_LOCKS>

    <DEFINE_LOCK Name="ITEM_LOC_LOCK">

        <SELECT>

            <PROPERTY Value="ITEM_ID"/>

            <PROPERTY Value="SHIP_TO_ID"/>

        </SELECT>

    </DEFINE_LOCK>

    <DEFINE_LOCK Name="ITEM_LOCK">

        <SELECT>

            <PROPERTY Value="ITEM_ID"/>

        </SELECT>

    </DEFINE_LOCK>
```

```
</DEFINE_LOCKS>
```

Each lock definition has a name that uniquely identifies the lock definition within a service.

The lock definition defines a list of select properties. This is the list of properties that will be concatenated together to form the lock key. The lock API will expect these properties in input and concatenation of those values will form the lock value. The format of the API is discussed below.

## Change service config file

Change the service config file to specify the lock definition file defined above.

```
<dom-config>
    <service-config Name="ORDER_ADMIN" TransactionCache="true"
Register="true">
        ...
        <dbLockSpecFiles>
            <dbLockSpecFile Name="xservice/orderadmin/xml/pce/dblock/
lockDefinition.xml"/>
        </dbLockSpecFiles>
        ...
    </service-config>
</dom-config>
```

## API specification

The API for acquiring a lock is shown below:

```
<ACQUIRE_DB_LOCK>
    <LOCKS>
        <LOCK Name="ITEM_LOC_LOCK"/>
    </LOCKS>
    <LOCK_DATA>
        <LOCK_UNIT> <!-- The API does not care for name of this tag.
It can be named anything -->
            <ITEM_ID Value="V-102-1"/>
            <SHIP_TO_ID Value="Velocity-Tonawanda"/>
        </LOCK_UNIT>
        <LOCK_UNIT> <!-- The API does not care for name of this tag.
It can be named anything -->
```

```
            <ITEM_ID Value="V-104-1"/>
            <SHIP_TO_ID Value="Velocity-Warren"/>
        </LOCK_UNIT>
    </LOCK_DATA>
</ACQUIRE_DB_LOCK>
```

The list of locks that need to be acquired is given under the `<LOCKS>` tag. Each lock is identified by the lock definition name that was defined in the lock definition file. Given the lock definition the API knows what data to look for under the `<LOCK_DATA>` tag.

The requested lock (in this case `ITEM_LOC_LOCK`) can be acquired for multiple data points (lock units) by specifying multiple `<LOCK_UNIT>` tags. In above case the `ITEM_LOC_LOCK` will be acquired for two lock units: `V-102-1^Velocity-Tonawanda^` and `V-104-1^Velocity-Warren^`.

In summary this API allows the user to identify the lock units and then lock them using predefined lock definitions.

The ACQUIRE_DB_LOCK API takes care of sorting all the lock units and inserting/ modifying the `LOCK_TABLE` in proper sequence so as to avoid any deadlocks. It has also been optimized to minimize the number of database calls that it does. If the transaction cache is turned on for the service then the API will batch the updates that it does.

# APPENDIX N Export Framework

### What's In This Appendix

This chapter gives you information on export framework features that have been implemented.

Topics include:


- Introduction
- Export Functionality
- Features of Export Framework
- Loading Export Definition
- Export Framework Schema


## Introduction

The following document lays out the detailed description of the export specification that can be used to specify export filters, document and its properties that have to be exported.

## Export Functionality

Exporting various documents/entities is an important functionality that the server needs to support. Examples of documents that may need this support are Purchase Orders, Shipments, Invoices, and Receipts etc. We need to export data from DOE to support various integration workflows and to help to generate various reports.

When export API is invoked, export framework selects the records which satisfy the export filters criteria and export them to the staging table which is defined in the same DB as the source table. After the data is exported, integration system can pickup the exported data and use them for their workflow. These exported data have to be purged periodically in order to get back the space used by them. You can use purge functionality provided by the export framework to accomplish this task.

## Features of Export Framework

The export framework makes it easier to define export filters on a document using its properties as well as the properties which are defined on its related documents. It also allows

you to define fields to be exported from the document and its related documents. For example, when receipt line is exported, you also want to export receipt date field which is defined on its header along with the receipt line fields.

## Key features of the export framework:

1. Ability to define which documents to export along with the attributes on the object itself as well as the related objects. It provides the ability where the external system which is triggering the export can specify the query parameters.
2. For example, schedules in a particular state, schedules for which the shipped qty >= ordered qty etc
3. Ability to store the exported document in the DB
4. Ability to export delta as well as whole. Where delta implies that the export is for only those objects that meet the selection criteria since the last export
5. Support for multiple external systems to export data simultaneously
6. Provide a log for the export activity which records the source system for the export trigger, the time when the export was performed, the API that was called, the time the job started and the time the job got completed and the number of records exported etc
7. Ability to specify number of export transactions you want to keep in the staging/export table
8. Support for locking to prevent concurrent export for the same spec and target system accidentally

## Steps involved in using export framework:

1. Define XML based export spec
2. Load export spec by specifying in the appropriate service configuration file
3. Create schema for the staging table which are needed to export the data
4. Invoke the export API to trigger exports

## Defining export definition

Export definition is a XML based specification and it captures all the data needed for the export framework to do its job. Please refer search framework to get more details on properties selection and filter definition etc since export framework is implemented using search framework libraries. It would be very clear if we define the export spec for the real problem.

### Requirements

Export shipment line which satisfies the following requirements:

- Fields to be exported from `SHIPMENT_LN` table are ID, CREATION_DATE, ITEM_DESCR, ITEM_ID, SHIPMENT_ID, SHIPPED_QTY, UOM
- Fields to be exported from related table `SHIPMENT` table are SELLING_ORG_ID, SHIP_TO_ID, SHIP_FROM_ID, SHIP_MODE, SHIP_DATE, and DELIVERY_DATE

- Support DELTA export based on the LAST_MODIFIED_DATE
- Export Shipment line which has at least one open PO line
- Export Shipment line for a given seller
- Export data into export table named EXP_ASN_LN

### Defining export spec consist the following steps:

1. Define export spec header
2. Define delta export filter
3. Define fields to be exported
4. Define fields which would be used in export filter

## Define export definition

This step provides helps to define export spec name, document to be exported, and name of the staging/export where you want to have the exported data

*Example:*

```
<export-defs>

<export-def

<!-- Name:  Export spec name -->

Name="ASN_EXPORT"

<!-- Document: Name of the document to be exported -->

Document="SHIPMENT_LINE"

<!-- ExportTableName:  Name of the export/staging table -->

ExportTableName="EXP_ASN_LN">

</export-def></export-defs>
```

## Define delta export filter

This step provides support to define delta export definition where you can specify filtering condition needed for the delta export using built-in variable "LAST_EXPORT_DATE" and "CURRENT_DATE". This filter expression would be converted into XPATH compliant filer expression by the export framework. The delta export filter is a fixed selection criterion and that can't be changed by the caller. It is strongly recommended to use "LAST_EXPORT_DATE" and "CURRENT_DATE" to set upper and lower bound for delta export. You can also specify the current date offset (in minutes) to prevent potential error cases due the transactions which are over lapping with export transaction. When export happens for the first time, "LAST_EXPORT_DATE" would be null since there is no earlier export. In this case "January 1, 1970" would be used as a last export date. Default value for "CurrentDateOffsetInMin" is 0.

By using "CURRENT_DATE" and current date offset, you can introduce a slack in the delta export. In the following example, delta filters restrict shipment line export i.e. it exports shipment lines which are modified since the last export and their modification date is less than or equal to (CURRENT_DATE - 5 minutes)

*Example:*

```
<export-defs>

<export-def Name="ASN_EXPORT" Document="SHIPMENT_LINE"
ExportTableName="EXP_ASN_LN">

<delta-export-filter CurrentDateOffsetInMin="5">

<AND>

 <LAST_MODIFIED_DATE MatchBy="GREATER" Value="LAST_EXPORT_DATE"/>

<LAST_MODIFIED_DATE MatchBy="LESS_EQUAL" Value="CURRENT_DATE"/>

</AND>

</delta-export-filter>

</export-def>

</export-defs>
```

## Define fields to be exported

This step provides support to define fields to be exported from the document and its related document. This feature gives you the control to select fields which are absolutely needed for your workflows.

You can select the properties from the related document which belongs to the same service as the document for which export spec is defined. Example when export spec is defined for SHIPMENT_LN which belongs to ORDER_ADMIN service, it is allowed to define properties from the related document such as SHIPMENT which also belongs to ORDER_ADMIN service but it is not allowed to use the related document say LOCATION which is defined in EM service. This limitation is imposed in order to create schema for the export/staging table dynamically

*Example:*

```
<export-defs>
   <export-def Name="ASN_EXPORT" Document="SHIPMENT_LINE"
   ExportTableName="EXP_ASN_LN">
      <select-properties>
         <!-- FIELDS FROM SHIPMENT_LN -->
         <ID/>
         <CREATION_DATE/>
         <ITEM_DESCR/>
         <ITEM_ID/>
```

```
            <SHIPMENT_ID/>
            <SHIPPED_QTY/>
            <UOM/>
            <!-- FIELDS FROM  RELATED DOCUMENT i.e. SHIPMENT (shipment
            header) -->
            <SELLING_ORG_ID MapTo="SHIPMENT_LINK/SELLING_ORG_ID"/>
            <SHIP_TO_ID MapTo="SHIPMENT_LINK/SHIP_TO_ID"/>
            <SHIP_FROM_ID MapTo="SHIPMENT_LINK/SHIP_FROM_ID"/>
            <SHIP_MODE MapTo="SHIPMENT_LINK/SHIP_MODE"/>
            <SHIP_DATE MapTo="SHIPMENT_LINK/SHIP_DATE"/>
            <DELIVERY_DATE MapTo="SHIPMENT_LINK/DELIVERY_DATE"/>
         </select-properties>
      </export-def>
   </export-defs>
```

### Define fields which would be used in export filter

This step provides support to define export filters using fields defined on the document to be exported and its related documents. The filter values would be specified by the caller at the run time

*Example:*

```
<export-defs>
   <export-def Name="ASN_EXPORT" Document="SHIPMENT_LINE"
   ExportTableName="EXP_ASN_LN">
      <!-- EXAMPLE FOR FILTER FIELD DEFINED ON SHIPMENT_LN -->
      <filter-def Name="CREATION_DATE" />
      <!-- EXAMPLE FOR FILTER FIELD DEFINED ON RELATED DOCUMENT i.e.
      SHIPMENT (shipment header) -->
      <filter-def Name="SELLING_ORG_ID"  MapTo="SHIPMENT_LINK/
      SELLING_ORG_ID"/>
   </export-def>
</export-defs>
```

# Loading Export Definition

The different export definitions that a service needs have to be specified in the service config xml file. The export definition files are defined in tag `<exportDefnFiles>`.

Service config file

```
<exportDefnFiles>
```

```
  <exportDefnFile Name="xservice/orderadmin/xml/pce/export/
  asnExportDef.xml"/>
  <exportDefnFile Name="xservice/orderadmin/xml/pce/export/
  openLineExportDef.xml"/>
</exportDefnFiles>
```

## Creating export/staging table schema

Export/Staging table scheme is derived from the export spec "<select-properties>" definition section. As you know, <select-properties> covers fields from the document to be exported as well as the fields from the related documents. This export/staging table schema would be registered to the XCORE when export spec is loaded by the XSERVICE. So when you run Schema Generator utility to generate schema, export/staging table schema is also generated. If you change the export spec which will affect the export/staging table schema then you need to recreate the scheme in order to keep export spec to be in sync with the physical schema.

## Invoking/Triggering export

You can use the newly created command "EXECUTE_EXPORT" to trigger export. In this command you can pass the export filter condition and can specify whether you need full export or delta export, target system name etc. Please make sure that you are using only those fields which are defined in the export spec filter definition section to define your filters.

It is assumed that export is not invoked for the same spec and target system with different export type (FULL, DELTA) i.e. first call export with export type as FULL and in the next time call export with export type DELTA. If the export type is changed between calls, you may get the wrong result due to not able to keep track of the `LAST_EXPORTED_DATE`.

*Example:*

```
<REQUESTS ServiceName="ORDER_ADMIN">

<EXECUTE_EXPORT

<!-- Name: Export spec name-->

Name="ASN_EXPORT"

<!--Type: Export type. Valid values = FULL|DELTA-->

 Type="DELTA"

<!--TargetSystem: Name of the system which is going to use the
exported data. Example: MRP, LEGACY, FP etc-->

TargetSystem="MRP"

<!--MaxPreviousExportToKeep: Number of export transaction to be kept
in DB    to support exported data recovery workflows. This will help
to purge unwanted exported data. Exported transaction count is
maintained for Export Spec Name, and Target System combination. If
you set this count as 5, when you export data for 6'th time then the
data exported as part of 1'st export would be purged and when you
```

```
export for the 7'th time then the data exported as part of 2'nd
export would be purged and so on. Set to 1 If you want to purge the
earlier exported data always before starting the new export-->

 MaxPreviousExportToKeep ="1">

 <!--FILTER: Export data filter; You can use filter which are
specified in the spec-->

<FILTER Name=" SELLING_ORG_ID">

<SELLING_ORG_ID Value="Saturn" MatchBy="EQUAL" />

</FILTER>

<FILTER Name="SCHEDULE_STATE_ID">

<OR>

<SCHEDULE_STATE_ID Value="ACCEPTED" MatchBy="EQUAL" />
<SCHEDULE_STATE_ID Value="IN_PROCESS" MatchBy="EQUAL" />

</OR>

</FILTER>

</EXECUTE_EXPORT>

</REQUESTS>

<RESPONSES Status="Success">

 <RESPONSE Status="Success">

<EXP_ID Value="EXPORT-3811099684440111"/>

<END_DATE Value="04/26/2004 00:01:36:000"/>

<EXP_TYPE Value="DELTA"/>

<NUM_RECORDS Value="12"/>

<PROGRESS Value="COMPLETED"/>

<PURGE_STATUS Value="NOT_PURGED"/>

<SPEC_NAME Value="OPEN_LINE_EXPORT"/>

<START_DATE Value="04/26/2004 00:01:36:000"/>

<TARGET_SYSTEM Value="MRP"/>

</RESPONSE>

</RESPONSES>
```

Since you have the capability to pass filters when you invoke export using "EXECUTE_EXPORT" command, if you want to export data for a specific period then you can achieve it by constructing appropriate export date range filter and then use it when you invoke the export command.

For each export spec and target system combination, last export date is maintained in `EXP_TRACKER` table. So when multiple target system requesting the delta exports using the same spec then export framework computes the delta using the data from `EXP_TRACKER` table to figure it out the delta for each target system independently.

# Export Framework Schema

Export framework persist export log and its details in the DB. Details are given below:

| SPEC_NAME | Spec name |
|---|---|
| TARGET_SYSTEM | Target system name |
| LST_EXP_DATE | Last export date and time |
| LST_EXP_ID | Last Export Id |
| SPEC_NAME | Spec name |
| TARGET_SYSTEM | Target system name |
| EXP_TYPE | Export type (Valid values: FULL, DELTA) |
| START_DATE | Start date and time |
| END_DATE | End date and time |
| NUM_RECORDS | Number of records exported |
| PROGRESS | Export progress status (Valid values: INITIATED, FAILED, and COMPLETED) |
| PURGE_STATUS | Purge status (Valid values: PURGED, NOT_PURGED) |

Additional field which would be added in each data export/staging table:

`EXP_ID_Export ID` (This field name ends with "_" in order to differentiate it from the exported fields)

To generate export framework schema and lock definition, you need to include export document definition file "com/teradata/xcore/export/export_doc.xml" and lock definition file "com/teradata/xcore/export/export_lock.xml" in xserver.xml.

*Example from xserver.xml:*

```
<docDefnFiles>

<docDefnFile Name="com/teradata/xcore/purge/purge_doc.xml"/>
<docDefnFile Name="com/teradata/xcore/export/export_doc.xml"/>

</docDefnFiles><dbLockSpecFiles>
```

```
<dbLockSpecFile Name="com/teradata/xcore/purge/purge_lock.xml"/>
<dbLockSpecFile Name="com/teradata/xcore/export/export_lock.xml"/>

</dbLockSpecFiles>
```

## Potential enhancements

1. Export data in CSV format. It would be pretty straight forward to export data from export/staging table to CSV format using DB native API's. This is the best approach since it would be pretty fast. If there is a need, we have to develop a utility class to export data from export/staging table into CSV format
1. Export data into XML format
2. Export into multiple export/staging table from an export spec. As of now, using an export spec, we can export data into only one export/staging table. This might be needed when you export RECEIPT_LINE and you also want to export the associated PO_LINE along with the RECEIPT_LINE. This enhancement would also be needed if you want to export the entire object as oppose to export a document. For example, export entire ASN which includes ASN, ASN_PKG, ASN_LINE, and ASN_LINE_PKG as part of one export. In the current scheme, you need to define multiple export specs to implement this requirement.
3. Provide UI support to search export transaction status

## Appendix 1

Sample export spec

```
<export-defs>

<export-def Name="ASN_EXPORT" Document="SHIPMENT_LINE"
ExportTableName="EXP_ASN_LN">

<delta-export-filter CurrentDateOffsetInMin="5">

<OR>

<!--using  shipment line's last modified date for delta export-->

<AND>

<LAST_MODIFIED_DATE MatchBy="GREATER" Value="LAST_EXPORT_DATE"/>

<LAST_MODIFIED_DATE MatchBy="LESS_EQUAL" Value="CURRENT_DATE"/>

</AND>    <!-- using  shipment header's last modified date for delta
export -->    <QUERY_DOCUMENT_LINK Name="SHIPMENT_LINK">

<AND>

<LAST_MODIFIED_DATE MatchBy="GREATER" Value="LAST_EXPORT_DATE"/>

<LAST_MODIFIED_DATE MatchBy="LESS_EQUAL" Value="CURRENT_DATE"/>

</AND>
```

```
</QUERY_DOCUMENT_LINK>

<!--using  order line's last modified date for delta export-->

<QUERY_DOCUMENT_LINK Name="SHIP_LN_TO_SCH_LINK">
<QUERY_DOCUMENT_LINK Name="SCHEDULE_LINK">

<AND>

<LAST_MODIFIED_DATE MatchBy="GREATER" Value="LAST_EXPORT_DATE"/>

<LAST_MODIFIED_DATE MatchBy="LESS_EQUAL" Value="CURRENT_DATE"/>

</AND>

</QUERY_DOCUMENT_LINK>

</QUERY_DOCUMENT_LINK>

</OR>

</delta-export-filter>

<select-properties>

<!-- fields from document to be exported -->

<ITEM_ID/>

<ITEM_DESCR/>

<REF_TYPE/>

<REF_ID/>

<UOM/>

<SHIPPED_QTY/>

<!-- fields from  related document -->

<SHIP_FROM_ID MapTo="SHIPMENT_LINK/SHIP_FROM_ID"/>     <SHIP_CHARGES
MapTo="SHIPMENT_LINK/SHIP_CHARGES"/>      <OTHER_CHARGES
MapTo="SHIPMENT_LINK/OTHER_CHARGES"/>      <GROSS_WEIGHT
MapTo="SHIPMENT_LINK/GROSS_WEIGHT"/>

<GW_UOM MapTo="SHIPMENT_LINK/GROSS_WEIGHT_UOM"/>      <SF_CNTRY_CODE
MapTo="SHIPMENT_LINK/SHIP_FROM_COUNTRY"/>       <SHIP_TERMS
MapTo="SHIPMENT_LINK/SHIP_TERMS"/>

<SHIP_VIA MapTo="SHIPMENT_LINK/SHIP_VIA"/>

<SHIP_MODE MapTo="SHIPMENT_LINK/SHIP_MODE"/>      <NUMBER_PACKAGES
MapTo="SHIPMENT_LINK/NUMBER_PACKAGES"/>      <TRACKING_NUMBER
MapTo="SHIPMENT_LINK/TRACKING_NUMBER"/>      <DELIVERY_DATE
MapTo="SHIPMENT_LINK/DELIVERY_DATE"/>      <SHIP_DATE
MapTo="SHIPMENT_LINK/SHIP_DATE"/>

</select-properties>
```

```
<!-- fields used in filter expression which comes from  document to
be exported -->

<filter-def Name="STATE_ID"/>

<filter-def Name="SHIPPED_QTY"/>

<!-- fields used in filter expression which comes from  related
document -->

<filter-def Name="SHIP_TERMS" MapTo="SHIPMENT_LINK/SHIP_TERMS"/>

<filter-def Name="SELL_ORG_ID" MapTo="SHIPMENT_LINK/SELLING_ORG_ID"/
>

<filter-def Name="SCH_STATE_ID" MapTo="SHIP_LN_TO_SCH_LINK/
SCHEDULE_LINK/STATE_ID"/>

<filter-def Name="DELIVERY_DATE" MapTo="SHIPMENT_LINK/
DELIVERY_DATE"/>

<filter-def Name="SHIP_TO_ID" MapTo="SHIPMENT_LINK/SHIP_TO_ID"/>

<filter-def Name="PLND_SHIP_DATE" MapTo="SHIPMENT_LINK/
PLANNED_SHIP_DATE"/>

<filter-def Name="PLND_DEL_DATE" MapTo="SHIPMENT_LINK/
PLANNED_DELIVERY_DATE"/>

</export-def>

</export-defs>
```

Sample export request and response:

```
<REQUESTS ServiceName="ORDER_ADMIN">

<EXECUTE_EXPORT

Name="ASN_EXPORT"

Type="DELTA"

TargetSystem="LEGACY"

MaxPreviousExportToKeep ="2">

<!--

<FILTER Name="STATE_ID">

<STATE_ID Value="SHIPPED" MatchBy="EQUAL" />

</FILTER>

<FILTER Name="SHIPPED_QTY">

<AND>

<SHIPPED_QTY Value="0" MatchBy="GREATER" />
```

```
<SHIPPED_QTY Value="100" MatchBy="LESS" />
</AND>
</FILTER>
-->
<FILTER Name="SHIP_TERMS">
<SHIP_TERMS Value="SHIP-01" MatchBy="EQUAL" />
</FILTER>
<FILTER Name="SELL_ORG_ID">
<SELL_ORG_ID Value="Saturn" MatchBy="EQUAL" />
</FILTER>
<!--
<FILTER Name="SCH_STATE_ID">
<OR>
<SCH_STATE_ID Value="ACCEPTED" MatchBy="EQUAL" />
<SCH_STATE_ID Value="IN_PROCESS" MatchBy="EQUAL" />
</OR>
</FILTER>
-->
<FILTER Name="DELIVERY_DATE">
<DELIVERY_DATE Value="03/20/2004" MatchBy="GREATER" />
</FILTER>
<FILTER Name="SHIP_TO_ID">
<SHIP_TO_ID Value="Velocity-Tonawanda" MatchBy="EQUAL" />
</FILTER>
<!--
<FILTER Name="PLND_SHIP_DATE">
<PLND_SHIP_DATE Value="03/20/2004" MatchBy="GREATER" />
</FILTER>
<FILTER Name="PLND_DEL_DATE">
<PLND_DEL_DATE Value="03/20/2004" MatchBy="GREATER" />
</FILTER>
-->
</EXECUTE_EXPORT>
```

```
</REQUESTS>
<RESPONSES Status="Success">
<RESPONSE Status="Success">
<EXP_ID Value="EXPORT-5181099960193744"/>
<END_DATE Value="04/26/2004 00:15:39:000"/>
<EXP_TYPE Value="DELTA"/>
<NUM_RECORDS Value="2"/>
<PROGRESS Value="COMPLETED"/>
<PURGE_STATUS Value="NOT_PURGED"/>
<SPEC_NAME Value="ASN_EXPORT"/>
<START_DATE Value="04/26/2004 00:15:39:000"/>
<TARGET_SYSTEM Value="LEGACY"/>
</RESPONSE>
</RESPONSES>
```

# APPENDIX O Purge Framework

### What's In This Appendix

This chapter gives you information onPurge framework and its features.

Topics include:


- Introduction
- Purge Functionality
- Features of Purge Framework
- Defining Purge Definition
- Loading Purge Definition
- Creating Purge Backup/Shadow Table Schema
- Invoking/Triggering Purge
- Purge Framework Schema
- Potential Enhancements
- Appendix 1


## Introduction

The following document lays out the detailed description of the purge specification that can be used to specify document, which needs to be purged.

## Purge Functionality

Purging documents/entities is an important functionality that the server needs to support. Examples of documents that may need this support are Purchase Orders, Shipments, Invoices, and Receipts etc. We need to purge unwanted data such as completed/cancelled/rejected orders, resolved alerts etc which are no more needed for our daily workflow. Keeping the unwanted data in database not only increase the space needed to maintain the database but also have an adverse effect on the performance of insert/delete/update database operations.

When purge API is invoked, purge framework selects the records which satisfy the purge criteria and purge them. If you want the purged data for reporting purpose then purge framework also provides the support to backup the purged data in shadow table defined in the same DB as the source table.

# Features of Purge Framework

The purge framework makes it easier to define document and related documents to be purged by allowing you to define the filtering criteria, dependency, traverse graph etc.

## Key Features of the Export Framework

- Pure Purging: Ability to delete objects from the database without storing this data elsewhere
- Purge and Copy: Copy existing documents based on certain criteria and saving them in shadow tables and then deleting the data from the original tables. The shadow tables mimic the schema of the copied documents in a separate table. These shadow tables are not accessible to the application and for all application purposes this data does not exist. The customer is responsible for maintaining these shadow tables
- Ability to define documents to be purged along with its related documents
- Ability to define purging criteria on the document to be purged and its related document
- Ability to specify shadow table names when you want to perform "Purge and Copy".
- Provide a log for the purge activity which records the spec name, the time the purge job started and the time the purge job got completed and the number of records purged etc
- Support for locking to prevent concurrent purge for the same spec accidentally

## Steps involved in using purge framework:

1. Define XML based purge spec
2. Load purge spec by specifying in the appropriate service configuration file
3. Create schema for the shadow table which are needed to backup the purge data if needed
4. Invoke the purge API to trigger purge task

# Defining Purge Definition

Purge definition is a XML based specification and it captures all the data needed for the purge framework to do its job. Following are the key capability it provides:

1. Ability to define documents to be purged
2. Ability to define purge criteria using the document to be purged and its related documents and to define purge criteria in terms of AGE using the built-in variable "AGE_IN_DAYS". Here you can specify purge criteria which are supported by XCORE
3. Ability define purge action such as PURGE, BACKUP_N_PURGE etc
4. Ability specify batch processing size to prevent out of memory exception
5. Ability to define the documents to be backed-up and its related documents
6. Ability to define the shadow table names if you want to backup the purged data

Simple purge definition (without related documents):

```
<purge_defs>

<!--Purge state history document whose creation date is older than
180 days-->

<purge_def

<!--Name: Purge spec name-->

Name="SHPurgeDefinition"

<!-- Action: Purge action. Valid values are PURGE and BACKUP_N_PURGE
-->

Action="PURGE"

<!-- AnchorDocumentName: Document to be purged -->
AnchorDocumentName="STATE_HISTORY"

<!-- BatchSize: Batch processing size which helps to prevent out of
memory exception when  you build     the graph using the related
documents-->
BatchSize="1000">

<documents>

<document DocumentName="STATE_HISTORY">

<!--filter: Condition to check the eligibility of the document and it
is expressed using standard xcore get doc filter format. You can
express it terms of AGE of the document using built-in variable
"AGE_IN_DAYS" -->

<filter>

<!--AGE_IN_DAYS: It will be translated into standard  xcore get doc
filter format-->

<AGE_IN_DAYS Value="180" FieldToBeUsed="CREATION_DATE"/>

</filter>

</document>

</documents>

</purge_def>

</purge_defs>
```

Complex purge definition (with related documents):

```
<purge_defs>

<!--

Back and then purge order schedule which satisifies the following
conditions:
```

```
StateId is in COMPLETED or REJECTED or CANCELLED state

Creation date is older than 180 days

NVOICE_STATUS of PO in which this order schedule belongs is in PAID
state

All the order schedules connected with the PO in which this order
schedules belongs should satisfy point number 1 to 4

-->

<purge_def

Name="POPurgeDefinition"

Action="BACKUP_N_PURGE"
AnchorDocumentName="ORDER_SCHEDULE"

BatchSize="250">

<!-- table_maps: Used to define the shadow table names    -->

<table_maps Prefix="A$_">

<table_map NativeName="ACTIVITY_ORDER" ArchiveName="AO"/>
<table_map NativeName="ORDER_SCHEDULE" ArchiveName="SCHEDULE"/>
<table_map NativeName="ORDER_NOTES" ArchiveName="NOTE"/>

</table_maps>

<documents>

<document DocumentName="ORDER_SCHEDULE">

<filter>

<AGE_IN_DAYS Value="300" FieldToBeUsed="CREATION_DATE"/>

<OR>

<STATE_ID Value="COMPLETED"/>

<STATE_ID Value="REJECTED"/>

<STATE_ID Value="CANCELLED"/>

</OR>

</filter>

</document>

<document DocumentName="ACTIVITY_ORDER">

<filter>

<INVOICE_STATUS Value="PAID"/>

</filter>

</document>
```

```
</documents>
<!-- dependency_links: Used to define eligibility criteria of the
document using the related documents -->
<dependency_links>
<dependency_link LinkName="AO_LINK">
<dependency_link LinkName="ORDER_SCHEDULES"/>
</dependency_link>
</dependency_links>
<!-- traverse_links: Used to back and delete related documents-->
<traverse_links>
<traverse_link LinkName="NOTES"/>
<traverse_link LinkName="AO_LINK">
<traverse_link LinkName="NOTES"/>
</traverse_link>
</traverse_links>
</purge_def>
</purge_defs>
```

## Assumptions

- Supports PURGE and BACKUP_N_PURGE functionality
- ServiceName at document level is an optional field. if not specified then name of the service which is loading purge_def would be used as a default
- Max row can be fetched is 100k
- BatchSize (default is 100) is an optional field which specifies what is the max number of anchor documents need to be processed in one batch
- Leaf node of the dependency graph should be the same as anchor document

*Example:*

```
<purge_defs>
<purge_def Name="CatTxnPurgeDefinition"   Action="BACKUP_N_PURGE"
AnchorDocumentName="ORDER_SCHEDULE"    BatchSize="100">
 ...
<dependency_links>
<dependency_link LinkName="AO_LINK">
<dependency_link LinkName="ORDER_SCHEDULES"/>
<!--  linked document is ORDER_SCHEDULE -->
```

```
</dependency_link>
...
</dependency_links>
</purge_def></purge_defs>
```

# Loading Purge Definition

The different purge definitions that a service needs have to be specified in the service config xml file. The purge definition files are defined in tag <purgeDefnFiles>

Service config file:

```
<purgeDefnFiles>
  <purgeDefnFile Name="xservice/orderadmin/xml/pce/purge/
  schPurgeDef.xml"/>
</purgeDefnFiles>
```

# Creating Purge Backup/Shadow Table Schema

Purge backup/shadow table scheme is identical to the source table expect the table. This backup/shadow table schema would be registered to the XCORE when purge spec is loaded by the XSERVICE. So when you run Schema Generator utility to generate schema, backup/shadow table schema is also generated. If you change the purge spec which will affect the backup/shadow table schema then you need to recreate the scheme in order to keep purge spec to be in sync with the physical schema.

# Invoking/Triggering Purge

You can use the newly created command "EXECUTE_PURGE" to trigger purge. In this command you specify the name of the purge spec.

## Invoke purge command

```
<REQUESTS ServiceName="ORDER_ADMIN">
  <EXECUTE_PURGE
  <!-- Name: Purge spec name-->
   Name="SHPurgeDefinition"/>
</REQUESTS>
<RESPONSES Status="Success">
```

```
  <RESPONSE BatchSize="3" AnchorDocumentsFetched="0"
  Status="Success">    <PURGE_ID Value="PURGE-3811099692974285"/>
     <END_DATE Value="04/26/2004 00:00:49:000"/>
     <NUM_RECORDS Value="0"/>
     <PROGRESS Value="COMPLETED"/>
     <SPEC_NAME Value="SHPurgeDefinition"/>
     <START_DATE Value="04/26/2004 00:00:49:000"/>
     <DETAILS>
     <STATE_HISTORY RowsAffected="0"/>
     </DETAILS>
  </RESPONSE>
</RESPONSES>
```

# Purge Framework Schema

Purge framework persist purge log and its details in the DB. Details are given below.

Table 101: PURGE_LOG Table

| PURGE_ID | Purge ID. System generated unique ID |
|---|---|
| SPEC_NAME | Purge spec name |
| START_DATE | Purge start date and time |
| END_DATE | Purge end date and time |
| NUM_RECORDS | Total Number of records purged including related documents |
| PROGRESS | Purge progress status (Valid values: INITIATED, FAILED, and COMPLETED) |

Table 102: PURGE_LOG_LN Table

| PURGE_ID | Purge ID |
|---|---|
| DOC_NAME | Document name. Example: ORDER_SCHEDULE, ACTIVITY_ORDER etc |
| NUM_RECORDS | Total Number of records purged from the document |

To generate purge framework schema and lock definition, you need to include purge document definition file "com/teradata/xcore/purge/purge_doc.xml" and lock definition file "com/teradata/xcore/purge/purge_lock.xml" in xserver.xml.

*Example from xserver.xml:*

```
<docDefnFiles>
<docDefnFile Name="com/teradata/xcore/purge/purge_doc.xml"/>
<docDefnFile Name="com/teradata/xcore/export/export_doc.xml"/>
</docDefnFiles>
<dbLockSpecFiles>
<dbLockSpecFile Name="com/teradata/xcore/purge/purge_lock.xml"/>
<dbLockSpecFile Name="com/teradata/xcore/export/export_lock.xml"/>
</dbLockSpecFiles>
```

# Potential Enhancements

1. Enable user to access archived data seamlessly. As of now after data is purged and backed-up/archived application can't access these data
2. Provide UI support to search purge transaction status
3. Copy static data when transaction data is copied: when transaction data is deleted from source table, the corresponding static data such as contacts, location, organization etc won't be deleted from the source table but the question is do you want to copy (not delete) the static data also along with transaction data to the backup table. Following are the few suggestions:
   - Copy static data along with transaction data and keep denormalized form of the transaction and the related static data in the same table i.e. in denormalized form
   - Copy static data along with transaction data and keep the transaction data in one table and the related static data in another shadow table with transaction object as the key

# Appendix 1

Sample purge spec:

```
<purge_defs>
<purge_def Name="SHPurgeDefinition"     Action="PURGE"
AnchorDocumentName="STATE_HISTORY"     BatchSize="250">
<documents>
 <document DocumentName="STATE_HISTORY">
<filter>
<AGE_IN_DAYS Value="180" FieldToBeUsed="DOC_DATE"/>
 </filter>
</document>
```

```
</documents>

</purge_def>

<purge_def Name="POPurgeDefinition"     Action="BACKUP_N_PURGE"
AnchorDocumentName="ORDER_SCHEDULE"     BatchSize="250">
<table_maps Prefix="A$_">

<table_map NativeName="ACTIVITY_ORDER" ArchiveName="AO"/>
<table_map NativeName="ORDER_SCHEDULE" ArchiveName="SCHEDULE"/>
<table_map NativeName="ORDER_NOTES" ArchiveName="NOTE"/>

</table_maps>

<documents>

<document DocumentName="ORDER_SCHEDULE">

<filter>

<AGE_IN_DAYS Value="300" FieldToBeUsed="CREATION_DATE"/>

</filter>

</document>

<document DocumentName="ACTIVITY_ORDER">

<filter>

<AGE_IN_DAYS Value="360" FieldToBeUsed="CREATION_DATE"/>

</filter>

</document>

</documents>

<dependency_links>

<dependency_link LinkName="AO_LINK">

<dependency_link LinkName="ORDER_SCHEDULES"/>

</dependency_link>

</dependency_links>

<traverse_links>

<traverse_link LinkName="NOTES"/>

<traverse_link LinkName="AO_LINK">

<traverse_link LinkName="NOTES"/>

</traverse_link>

</traverse_links>

</purge_def>

</purge_defs>
```

# APPENDIX P Search Framework

### What's In This Appendix

This chapter gives you information onSearch Framework.

Topics include:


- Introduction
- Search Functionality
- Search Definition
- Interfaces to Search Framework
- Loading the Search Definition
- Appendix 1


## Introduction

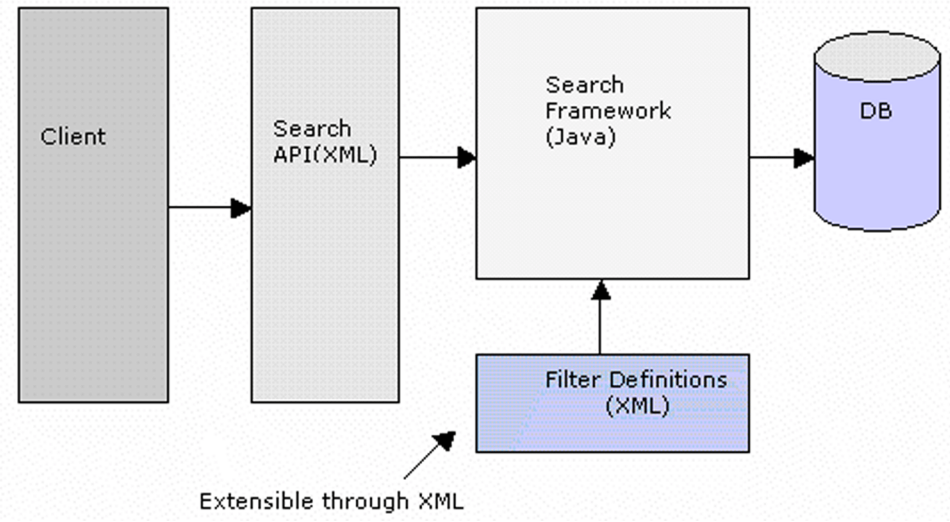
The following document lays out the detailed description of the search filter specification that can be used to support different filters on a document.

## Search Functionality

Searching for different documents/entities is an important functionality that the server needs to support. Examples of documents that may need this support are Customer Orders, Shipments, Invoices, OrderPoints etc. The search framework makes is easy to implement search capability on a document defined in xservices. It allows a document to be searched by properties that are defined for the document as well as search on properties on linked documents. E.g. searching Customer Orders by ItemId, which is a property of the lines of the order.

The following diagram shows various parts of the existing search framework:

Figure 104: Search Framework




The request and response format for a typical search API is:

```
<REQUEST Name="getCustomerOrders" StartAtRow="10" MaxRows="20"
ReturnRowCount="yes" UserId="bbabin">

<FILTER Name="ID">

<OR>

<ID Value="CO-1" MatchBy="STARTS_WITH"/>

<ID Value="123" MatchBy="ENDS_WITH"/>

</OR>

</FILTER>

<FILTER Name="CREATION_DATE">

<CREATION_DATE Value="12/12/2001" MatchBy="GREATER"/>
<CREATION_DATE Value="12/15/2001" MatchBy="LESS"/>

</FILTER>

<FILTER Name="ORDER_POINT_ID">

<ORDER_POINT_ID Value="OP-1"/>

</FILTER>

    ... Other filters as defined in the search definition ...

<ORDER_BY>

<CREATION_DATE Sort="Ascending"/>

<ID Sort="Descending"/>

</ORDER_BY>
```

```
</REQUEST>

<RESPONSE>

<CUSTOMER_ORDERS TotalRowCount="12">

<CUSTOMER_ORDER ExistingDocument="yes" >

<ID Value="CO_1" />

<ORDER_ORIGIN_ID Value="CSR" />

<ORDER_POINT_ID Value="OP_011" />

 ... other properties ...

</CUSTOMER_ORDER>

<CUSTOMER_ORDER ExistingDocument="yes" >

<ID Value="CO_1" />

<ORDER_ORIGIN_ID Value="CSR" />

<ORDER_POINT_ID Value="OP_011" />

... other properties ...

</CUSTOMER_ORDER>

 ... other customer orders ...

 </CUSTOMER_ORDERS>

</RESPONSE>
```

For the request if the attributes StartAtRow and MaxRows are present then the server will do pagination on the result. The ReturnRowCount attribute is optional. If this attribute value is 'yes' then the TotalRowCount is returned as part of the response. The UserId attribute is mandatory only if the search involves domain check.

The response will contain the documents that matched given search criteria. The result may be paginated depending on input request.

The filters that are supported depend on the search definition that is used. Search definition is where you can specify the filters and how they map to in the query.

# Search Definition

Search definition is a specification that tells the search framework how to generate the database query.

Search Definition Example

```
<search-defs>

<search-def Name="SIMPLE_CO_SEARCH" Document="CUSTOMER_ORDER">
```

```
<select-properties>
<ID/><ORDER_POINT_ID/>
<CREATION_DATE/>
<CREATED_BY_ID/>
<NAME/>
<DOC_TYPE/>
<ORDER_TYPE_ID/>
<STATE_ID/>
<ESTMT_VALUE/>
<CURRENCY_CODE/>
<PRICE MapTo="ORDER_LINE_ITEMS/UNIT_PRICE"/>
<REQUESTE_DATE MapTo="ORDER_LINE_ITEMS/ORDER_REQUESTS/
DELIVERY_DATE"/>
<PROMISE_DATE MapTo="ORDER_LINE_ITEMS/ORDER_REQUESTS/
ORDER_SCHEDULES/DELIVERY_DATE"/>
<ITEM_ID MapTo="ORDER_LINE_ITEMS/ITEM_ID"/>
</select-properties>
<fetch-document-link Name="NOTES">
<select-properties>
<NOTE_TYPE_ID/>
</select-properties>
</fetch-document-link>
<filter-def Name="ID"/>
<filter-def Name="ORDER_TYPE_ID"/>
<filter-def Name="CREATION_DATE"/>
<filter-def Name="STATE_ID"/>
<filter-def Name="ESTMT_VALUE"/>
<filter-def Name="EXT_ORDER_NUMBER" MapTo="EXT_ORDER_NUM"/>
<filter-def Name="ITEM_ID" MapTo="ORDER_LINE_ITEMS/ITEM_ID"/>
<domain-filter-defs RuleName="getDomains">
<domain-filter-def Name="LOCAL_ORDER_POINTS"
EmResponse="ORDER_POINT_ID" MapTo="ORDER_POINT_ID"
MapToQueryLink="LOCAL_ORDER_POINT_LINK">
<em-filter-def Name="ORDER_POINT_NAME" EmFilterName="NAME"/>
```

```
<em-filter-def Name="ORDER_POINT_ID" EmFilterName="ORDER_POINT_ID"/>

</domain-filter-def></domain-filter-defs>

<rule-filter-def Name="CUSTOMER_ORG_ID"
RuleName="getOpIdsFromOrgIds" MapTo="ORDER_POINT_ID"
MapToQueryLink="ORDER_POINT_LINK"/>

</search-def></search-defs>
```

Each search definition is defined for a document and should have a unique name. In above example the search definition name is '`SIMPLE_CO_SEARCH`' and it's defined for document '`CUSTOMER_ORDER`'.

The search definition allows users to define the fields that are to be selected and retuned in the response as well as other associated documents that are to be fetched.

The search definition also allows the user to define various filters that will be supported in the search API. There are three types of filters: Normal Filters, Domain Filters and Rule Filters. These are described in the following sections

## Selected properties

The `<select-properties>` tag is used to define the fields that need to be returned as part of the search response. The fields listed under this tag can belong to the document for which the search spec is defined as well as other documents linked to the primary document.

The fields belonging to the document are listed by just the names. The fields of the linked documents are listed with an alias and a mapping path to access the linked document field.

In the example above, `REQUEST_DATE` has a '`MapTo`' defined as below.

```
ORDER_LINE_ITEMS/ORDER_REQUESTS/DELIVERY_DATE
```

The system will perform a join of the primary document `CUSTOMER_ORDER` and the linked documents `ORDER_LINE_ITEM` and `ORDER_REQUEST` to get the field `DELIVERY_DATE` defined for `ORDER_REQUEST` document.

The selected properties are returned in the response as first-class citizen properties of each returned record.

See “Appendix 1” in this chapter for an example of an actual search query and response.

## Fetch document links

The `<fetch-document-link Name="ABC">` tag defines the associated documents that need to be fetched and returned as part of the response.

See “Appendix 1” in this chapter for an example of an actual search query and response.

## Normal filters

*Example:*

```
<filter-def Name="EXT_ORDER_NUMBER" MapTo="EXT_ORDER_NUM"/>
```

```
<filter-def Name="ITEM_ID" MapTo="ORDER_LINE_ITEMS/ITEM_ID"
CaseInsensitive="true" />
```

The <filter-def> tag defines a normal filter. A normal filter is a filter that does not need any special processing and is evaluated by converting to a GET_DOCUMENT call. The 'Name' attribute is a mandatory attribute. This attribute defines what the input request filter name should be.

The input filter name need not be same as the required document property name. The 'MapTo' attribute can be used to define how the input filter should be mapped to database. e.g. above the input filter name 'EXT_ORDER_NUMBER' is mapped to 'EXT_ORDER_NUM' in the document definition (i.e. the CUSTOMER_ORDER document has a property called EXT_ORDER_NUM).

It's also possible that the filter may map to a property on some linked document. E.g. the 'ITEM_ID' filter is mapped to document 'ORDER_LINE_ITEM' and property 'ITEM_ID' through link 'ORDER_LINE_ITEMS'.

A string search by default is case sensitive. To perform a case insensitive search, users have to specify the attribute CaseInsensitive="true". If the attribute is set to true, a case insensitive search will be performed on the filter property.

## Domain filters

Example:

```
<domain-filter-defs RuleName="getDomains">

<domain-filter-def Name="LOCAL_ORDER_POINTS"
EmResponse="ORDER_POINT_ID"
MapTo="ORDER_POINT_ID"
MapToQueryLink="LOCAL_ORDER_POINT_LINK">

<em-filter-def Name="ORDER_POINT_NAME" EmFilterName="NAME"/><em-
filter-def Name="ORDER_POINT_ID" EmFilterName="ORDER_POINT_ID"/>

</domain-filter-def>

</domain-filter-defs>
```

The <domain-filter-defs> tag defines the domain filters. There can be multiple domain filters. The <domain-filter-def> tag defines each domain filter. The domain filter is only needed to filter by user domains. By default, the user domains are assumed to be specified in EM service. If there is a new service that maintains domains conforming to the sane domain API, search framework can invoke that service instead of EM. The service name for such a service can be specified as the value of service parameter named SEARCH_DOMAIN_SERVICE.

Refer to xcore.doc for documentation on setting up a service parameter.

The EM service allows domain setup for each user. As example is: a user having role of CSR and having access to a list of customers. Now if this user searches for customer orders he should be able to see only those orders where the order's order-point belongs to one of the enabled customers. This is handled by defining the domain filters.

The domain filters are automatically applied if they are defined in the search definition. They don't need to be specified as part of the search API. If the input UserId is null then the domain filters are not applied.

In the definition the '`RuleName`' attribute on `<domain-filter-defs>` tag defines the rule in EM that is called to get the user domain information. The value of this attribute is 'getDomains' and it does not need to change across implementation.

The '`Name`' attribute on '`domain-filter-def`' tag specifies the domain name as defined by EM service. The various domains supported by EM service today are:

- 
- CUSTOMER_ORGS
- SELLING_ORGS
- FULFILLMENT_CENTERS
- LOCATIONS
- LOCAL_ORDER_POINTS
- LOCAL_BILL_TOS
- LOCAL_SHIP_TOS

Have a look at EM documentation for 'getDomains' API for an updated list of supported domains.

What the 'getDomains' API returns is a list of entities that the user has access to or xml query that can be executed to get this list. The format of EM API request and response are:

EM API Response

```
<REQUEST Name="getDomains">

<USER_ID Value="user3"/>

<FILTER Name="STATUS">

<OR>

<STATUS Value="ACTIVE"/>

<STATUS Value="DORMANT"/>

</OR>

</FILTER>

<DOMAINS>

<LOCAL_ORDER_POINTS>

<RETURN_XML_QUERY Value="yes"/>

 <!-- This is sent as 'yes' if MapToQueryLink attribute is defined on
the domain-filter-def tag -->

</LOCAL_ORDER_POINTS>

</DOMAINS>
```

```
</REQUEST>

<!-- Response in case where RETURN_XML_QUERY value in request to EM
is 'yes' -->

<RESPONSE Status="Success">

<DOMAINS>

<LOCAL_ORDER_POINTS>

<GET_DOCUMENT Name="LOCAL_ORDER_POINT" RemoveEmptyQdl="yes"
AssignToVar="eopDocs">

<SELLING_ORG_ID Value="ORG_1"/>

<OR><STATUS Value="ACTIVE"/>

<STATUS Value="DORMANT"/>

</OR>

<QUERY_DOCUMENT_LINK Name="URDM_CUST_LINK">

<DOMAIN_TYPE_ID Value="CUST"/>

<URM_ID Value="URM-1003"/>

</QUERY_DOCUMENT_LINK>

<OR>

<OP_STATUS Value="ACTIVE"/>

<OP_STATUS Value="DORMANT"/>

</OR>

</GET_DOCUMENT>

</LOCAL_ORDER_POINTS>

 … other domains …

</DOMAINS><

/RESPONSE>
```

or

```
<!-- Response in case where RETURN_XML_QUERY value in request to EM
is 'no' -->

<RESPONSE Status="Success">

<DOMAINS>

<CUSTOMER_ORGS>

<ID Value="org-1"/>

<ID Value="org-2"/>
```

```
... more ID tags ...
</CUSTOMER_ORGS>
<LOCAL_ORDER_POINTS>
<LOCAL_ORDER_POINT>
<ORDER_POINT_ID Value="op01"/>
<SELLING_ORG_ID Value="org01"/>
</LOCAL_ORDER_POINT>
<LOCAL_ORDER_POINT>
<ORDER_POINT_ID Value="op02"/>
<SELLING_ORG_ID Value="org01"/>
</LOCAL_ORDER_POINT>.
.. more LOCAL_ORDER_POINT tags ...
</LOCAL_ORDER_POINTS>.
.. other domains ...
</DOMAINS>
</RESPONSE>
```

The '`MapTo`' and '`MapToQueryLink`' attributes are used to specify how the response that EM returned should be used in filtering the search results.

The '`MapTo`' attribute specifies how the domains returned by EM map to in relation to search document. In above example by specifying the `MapTo="ORDER_POINT_ID"` we are saying that the ORDER_POINT_IDs returned by EM in response of 'getDomains' API should be mapped to ORDER_POINT_ID field on CUSTOMER_ORDER document i.e return only those CUSTOMER_ORDER documents whose ORDER_POINT_ID field value is present in the list returned by EM service.

The '`MapTo`' attribute can also specify a path if the mapped property is not part of the given document. If you want to search for ACTIVITY_ORDER and want ORDER_POINT_ID domain then the 'MapTo' attribute will look like: `MapTo="CO_LINK/ORDER_POINT_ID"` where '`CO_LINK`' is a document link from ACTIVITY_ORDER document to CUSTOMER_ORDER document.

The '`EmResponse`' attribute on the `<domain-filter-def>` tag is used to specify what property to pick from EM's response in cases where the data is coming in a nested format. For example as shown above for the LOCAL_… domains EM returns the data in a nested structure. This attribute should not be specified for domains like CUSTOMER_ORGS where the response is just a list of IDs.

**Note:** The '`MapTo`' attribute is in process of being deprecated for performance reasons and its use should be avoided. Instead the '`MapToQueryLink`' attribute should be used as explained below.

When the 'MapToQueryLink' attribute is specified instead of getting a list of valid domain entities from EM service the search framework will get the query that EM would have executed and makes it a sub-query of the search query itself i.e. it will do a join across service to EM documents.

The value of 'MapToQueryLink' attribute specifies name of the document link that links the search document i.e. CUSTOMER_ORDER in this example to the EM domain document i.e. LOCAL_ORDER_POINT for the LOCAL_ORDER_POINTS domain. The search initialization will fail at load time if this link is not defined in the document definition. In above example the link name is "LOCAL_ORDER_POINT_LINK".

In above example the final query that search framework will generate after taking domains into consideration will look like:

Result of domain check on search query:

```
<GET_DOCUMENT Name="CUSTOMER_ORDER" StartAtRow="0" MaxRows="10"
ReturnRowCount="yes">

... conditions based on filters specified ...

<QUERY_DOCUMENT_LINK Name="LOCAL_ORDER_POINT_LINK">

<!-- Domain check done by using the xml query returned by EM service-
->

<SELLING_ORG_ID Value="ORG_1"/><OR><STATUS Value="ACTIVE"/>

<STATUS Value="DORMANT"/>

</OR>

<QUERY_DOCUMENT_LINK Name="URDM_CUST_LINK">

<DOMAIN_TYPE_ID Value="CUST"/>

<URM_ID Value="URM-1003"/>

</QUERY_DOCUMENT_LINK>

<OR>

<OP_STATUS Value="ACTIVE"/>

<OP_STATUS Value="DORMANT"/>

</OR>

</QUERY_DOCUMENT_LINK>

</GET_DOCUMENT>
```

Valid EM document for each domain is as follows:

| Domain | Document |
|---|---|
| CUSTOMER_ORGS | ORGANIZATION |

| Domain | Document |
|---|---|
| SELLING_ORGS | ORGANIZATION |
| FULFILLMENT_CENTERS | LOCATION |
| LOCATIONS | LOCATION |
| LOCAL_ORDER_POINTS | LOCAL_ORDER_POINT |
| LOCAL_BILL_TOS | LOCAL_BILL_TO |
| LOCAL_SHIP_TOS | LOCAL_SHIP_TO |

The search API may also need to support filters on the domain info that EM returns. E.g. ORDER_POINT_NAME is a filter for order points, which is the domain for Customer Orders. Such filters are handled by the EM service and the search framework just has to do tag name conversions and send the input filters to EM.

The '`em-filter-def`' tag defines such filters. In above example the input will support filters 'ORDER_POINT_NAME' and 'ORDER_POINT_ID' and these filters are sent to EM service for evaluation. The '`EmFilterName`' attribute specifies the filter name that EM service understands. Check EM service documentation for 'getDomains' API to find valid EM filter names.

## Rule filters

Example:

```
<rule-filter-def Name="CUSTOMER_ORG_ID"
RuleName="getOpIdsFromOrgIds" MapTo="ORDER_POINT_ID"
MapToQueryLink="ORDER_POINT_LINK"/>
```

The `<rule-filter-def>` tag defines a rule filter. A rule filter is used when the input filter needs conversion before it can be applied. In above example the input filter is CUSTOMER_ORG_ID. The CUSTOMER_ORDER document does not have CUSTOMER_ORG_ID field but has ORDER_POINT_ID field. So before the CUSTOMER_ORG_ID filter can be applied it has to be converted to a valid list of ORDER_POINT_IDs. This is done using the rule specified by the 'RuleName' attribute.

The resulting ORDER_POINT_Ids are then used to search CUSTOMER_ORDER.

The use of '`MapTo`' and '`MapToQueryLink`' attributes in `<rule-filter-def>` is similar to their use in `<domain-filter-def>` tag.

The API for the rule specified in '`RuleName`' is:

```
<!-- The request filter tag is the filter tag as received in input
with the filter name converted to value specified by 'RuleFilterName'
attribute in the filter definition, if theattribute is specified.-->

<REQUEST Name="ruleName">

<USER_ID Value="user1"/>
```

```
 <FILTER Name="ORG_ID">
<!-- filter name as defined by the Name attribute of rule-filter-def
-->
<OR>
<ORG_ID Value="15%" MatchBy="STARTS_WITH"/>
<!-- filter data as given by the user in input -->
 <ORG_ID .../>
</OR>
</FILTER>
<RETURN_XML_QUERY Value="yes"/>
</REQUEST>Expected Response if RETURN_XML_QUERY in input is 'yes':
<RESPONSE>
<GET_DOCUMENT Name="ORDER_POINT">
... condition to get valid data considering the given filter and
USER_ID ...
</GET_DOCUMENT>
</RESPONSE>
```

Expected Response if `RETURN_XML_QUERY` in input is '`no`':

```
<RESPONSE>
<FILTER Name="ORG_ID">
<!-- Name attribute value here does not matter -->
<OR>
<!-- The response should always have an OR tag -->
<ID Value="AO-1"/>
<!-- Name of this tag does not matter -->
<ID Value="AO-56"/>
</OR>
</FILTER>
</RESPONSE>
```

The rule '`getOpIdsFromOrgIds`' mentioned above looks like:

**`Rule getOpIdsFromOrgIds`**

The request to this rule will look like:

```
<REQUEST Name="getOpIdsFromOrgIds">
```

```
<USER_ID Value="user6"/>

<FILTER Name="CUSTOMER_ORG_ID">

<OR>

<CUSTOMER_ORG_ID Value="12%" MatchBy="STARTS_WITH"/><CUSTOMER_ORG_ID
Value="ORG_1"/>

</OR>

</FILTER>

<RETURN_XML_QUERY Value="yes"/>

</REQUEST>
```

The response from this rule if `RETURN_XML_QUERY` is set to '`yes`' should look like:

```
<RESPONSE>

<GET_DOCUMENT Name="ORDER_POINT">.

.. condition to get valid data considering the given filter and
USER_ID. This particular rule call EM to get this ...

</GET_DOCUMENT>

</RESPONSE>
```

The response from this rule if `RETURN_XML_QUERY` is set to '`no`' will look like:`<RESPONSE>`

```
<FILTER Name="ORDER_POINT_ID">

<OR>

<ID Value="AO-1"/>

<ID Value="AO-56"/>

</OR></FILTER>

</RESPONSE>
```

The rule definition looks like:

```
<DEFINE_METHOD Name="getOpIdsFromOrgIds">

<RULE>

<ACTION>

<IF_TEST Test="$thisParam/RETURN_XML_QUERY/@Value != 'yes'">

<THEN>

<REQUEST Name="getOrderPoints" AssignToVar="result"
ServiceName="EM"><USER_ID Select="$thisParam/USER_ID/@Value"/>

<TO_XML SelectList="$thisParam/FILTER"/>

</REQUEST>
```

```
<IF_TEST Test="$result/@Status = 'Success'">

<THEN>

<TO_DOCVAR AssignToVar="filterDoc">

<FILTER Name="ORDER_POINT_ID"/>

</TO_DOCVAR>

<TO_DOCVAR AssignToVar="orFilterDoc">

<OR/>

</TO_DOCVAR><ADD_CHILD ChildDocVar="orFilterDoc" DocVar="filterDoc"/
><FOR_EACH SelectList="$result/ORDER_POINTS/ORDER_POINT"
CurrentElement="opDoc">

<SET Var="id" FromSelect="$opDoc/ID/@Value"/>

<TO_DOCVAR AssignToVar="idFilterDoc">

<ID Var="id"/>

</TO_DOCVAR>

<ADD_CHILD ChildDocVar="idFilterDoc" DocVar="orFilterDoc"/>

</FOR_EACH>

</THEN>

</IF_TEST>

<ADD_CHILD ChildDocVar="filterDoc" DocVar="thisReturn"/>

</THEN>

<ELSE>

<REQUEST Name="getOrderPoints" ServiceName="EM"
AssignToVar="responseForm">

<USER_ID Select="$thisParam/USER_ID/@Value"/>

<TO_XML SelectList="$thisParam/FILTER"/>

<RETURN_XML_QUERY Value="yes"/>

</REQUEST>

<APPEND_CHILDREN DocVar="thisReturn" FromDocVar="responseForm"/>

</ELSE>

</IF_TEST>

</ACTION>

</RULE>

</DEFINE_METHOD>
```

## Specifying additional search criteria

In certain cases, there may be a need to specify additional attributes or XML tags in the GET_DOCUMENT that search framework generates for a search request. To be able to add these additional attributes and append XML tags, one can specify a get-doc-spec in the search spec like below. The specified attributes and XML in the get-doc-spec will be transferred to the get document XML generated by the search framework.

Using get-doc-spec to specify search rules:

```
<search-def Name="CUSTOMER_ORDER_SEARCH" Document="CUSTOMER_ORDER">

<get-doc-spec>

<!-- This will add all the specified attributes to the get form that search builds. Note that these attribute values will override whatever was computed by search spec. So, one needs to know what one is doing when using these.-->

<addAttr Name="Distinct" Value="false"/>

<addAttr Name="AnyAttrName" Value="AnyAttrValue"/>

<appendXml>

<XCORE_HINT>

<SELECT_QUERY Type="Subquery"/>

<!-- The other tag vaiations supported are :

<SELECT_QUERY Type="InlineJoin"/>

<COUNT_QUERY Type="InlineJoin"/>
```

or

```
<COUNT_QUERY Type="Subquery"/> -->

</XCORE_HINT>

</appendXml>

</get-doc-spec>

<select-properties>

<ID/>

<ORDER_POINT_ID/>

</select-properties>

….Search spec details

</search-def>
```

# Interfaces to Search Framework

Search specification can be used for searching using an xcommand or xrule interface.

XCommand interface to search framework:

```
<REQUESTS ServiceName="ORDER_ADMIN">

<!-- GET_SEARCH_RESULTS will return the results of a search -->
<GET_SEARCH_RESULTS>

<SPEC Value="CUSTOMER_ORDER_SEARCH"/>

<CRITERIA>

<REQUEST Name="getCustomerOrders" MaxRows="10" ReturnRowCount="yes"
StartAtRow="0" name="//OrderAdminService/getCustomerOrders"
UserId="user3">

<FILTER Name="NAME"/>

<FILTER Name="ID">

<ID Value="" MatchBy="ENDS_WITH"/>

</FILTER>

<FILTER Name="CREATION_DATE"/>

<FILTER Name="ORDER_POINT_ID"/>

<FILTER Name="ORDER_POINT_NAME"/>

<FILTER Name="ORDER_TYPE_ID">

<ORDER_TYPE_ID Value="STANDARD" MatchBy="EQUAL"/>

</FILTER><FILTER Name="STATE_ID">

<STATE_ID Value="INITIATED" MatchBy="NOT_EQUAL"/>

</FILTER><FILTER Name="DOC_TYPE">

<DOC_TYPE Value="ORDER"/>

</FILTER>

<ORDER_BY>

<CREATION_DATE Sort="Ascending"/>

</ORDER_BY>

</REQUEST>

</CRITERIA>

<ROOT_NAME Value="CUSTOMER_ORDERS"/>

</GET_SEARCH_RESULTS>
```

```
<!-- GET_SEARCH_FORM will return the GET_DOCUMENT query that is
composed to do a search -->

<GET_SEARCH_FORM>

<SPEC Value="CUSTOMER_ORDER_SEARCH"/>

<CRITERIA>

<REQUEST Name="getCustomerOrders" MaxRows="10" ReturnRowCount="yes"
StartAtRow="0" name="//OrderAdminService/getCustomerOrders"
UserId="user3">

<FILTER Name="NAME"/>

<FILTER Name="ID">

<ID Value="" MatchBy="ENDS_WITH"/>

</FILTER>

<FILTER Name="CREATION_DATE"/>

<FILTER Name="ORDER_POINT_ID"/>

<FILTER Name="ORDER_POINT_NAME"/>

<FILTER Name="ORDER_TYPE_ID">

<ORDER_TYPE_ID Value="STANDARD" MatchBy="EQUAL"/>

</FILTER>

<FILTER Name="STATE_ID">

<STATE_ID Value="INITIATED" MatchBy="NOT_EQUAL"/>

</FILTER>

<FILTER Name="DOC_TYPE"><DOC_TYPE Value="ORDER"/>

</FILTER>

<ORDER_BY><CREATION_DATE Sort="Ascending"/>

</ORDER_BY>

</REQUEST>

</CRITERIA>

<ROOT_NAME Value="CUSTOMER_ORDERS"/>

</GET_SEARCH_FORM></REQUESTS>The response to the above request will
be of the format<RESPONSES Status="Success">

<!-Response to GET_SEARCH_RESULTSà

<RESPONSE Status="Success">

<CUSTOMER_ORDERS TotalRowCount="1" Status="Success">

<CUSTOMER_ORDER ExistingDocument="yes">
```

```
<ID Value="CO-1"/>

<ORDER_POINT_ID Value="OP_1"/>

<CREATION_DATE Value="07/22/2004 09:28:15:000"/>

<CREATED_BY_ID Value="user3"/>

<DOC_TYPE Value="ORDER"/>

<ORDER_TYPE_ID Value="STANDARD"/>

<STATE_ID Value="SENT_FOR_FULFILLMENT"/>

<ESTMT_VALUE Value="59554.11"/>

<CURRENCY_CODE Value="USD"/>

<BILL_TO_ID Value="BT_1"/>

<SHIP_TO_ID Value="SP_1"/>

</CUSTOMER_ORDER>

</CUSTOMER_ORDERS>

</RESPONSE>

<!-Response to GET_SEARCH_FORMà<RESPONSE Status="Success">

<GET_DOCUMENT Name="CUSTOMER_ORDER" StartAtRow="0" MaxRows="10"
ReturnRowCount="yes">

<SELECT>

<ID/>

<ORDER_POINT_ID/>

<CREATION_DATE/>

<CREATED_BY_ID/>

<NAME/>

<DOC_TYPE/>

<ORDER_TYPE_ID/>

<STATE_ID/>

<ESTMT_VALUE/><CURRENCY_CODE/>

<BILL_TO_ID/>

<SHIP_TO_ID/></SELECT>

<ORDER_BY>

<CREATION_DATE Sort="Ascending"/>

</ORDER_BY><ID Value="" MatchBy="ENDS_WITH"/>

<ORDER_TYPE_ID Value="STANDARD" MatchBy="EQUAL"/>
```

```
<STATE_ID Value="INITIATED" MatchBy="NOT_EQUAL"/>
<DOC_TYPE Value="ORDER"/>
<QUERY_DOCUMENT_LINK Name="LOCAL_ORDER_POINT_LINK"><SELLING_ORG_ID
Value="ORG_1"/>
<OR>
<STATUS Value="ACTIVE"/>
<STATUS Value="DORMANT"/>
</OR>
<QUERY_DOCUMENT_LINK Name="URDM_CUST_LINK">
<DOMAIN_TYPE_ID Value="CUST"/>
<URM_ID Value="URM-1003"/>
</QUERY_DOCUMENT_LINK>
<OR>
<OP_STATUS Value="ACTIVE"/>
<OP_STATUS Value="DORMANT"/>
</OR>
</QUERY_DOCUMENT_LINK>
</GET_DOCUMENT>
</RESPONSE></RESPONSES>
```

XRule interface to search framework:

```
<DEFINE_METHOD Name="getCustomerOrders">
<RULE>
<ACTION>
<GET_SEARCH_RESULTS AssignToVar="thisReturn">
<SPEC Value="CUSTOMER_ORDER_SEARCH"/>
<CRITERIA>
<!-- 'thisParam' has the input to the selection criteria as specified
in the XML under CRITERIA tag in XCommand interface example-->
<TO_XML DocVar="thisParam"/>
</CRITERIA>
<!-- ROOT_NAME is an optional tag-->
<ROOT_NAME Value="CUSTOMER_ORDERS"/>
</GET_SEARCH_RESULTS>
```

```
</ACTION>

</RULE>

</DEFINE_METHOD>
```

Upon execution, the 'thisReturn' variable points to following XML <RESPONSE Status="Success">:

```
<CUSTOMER_ORDERS TotalRowCount="1" Status="Success"><CUSTOMER_ORDER
ExistingDocument="yes">

<ID Value="CO-1"/>

<ORDER_POINT_ID Value="OP_1"/>

<CREATION_DATE Value="07/22/2004 09:28:15:000"/>

<CREATED_BY_ID Value="user3"/>

<DOC_TYPE Value="ORDER"/>

<ORDER_TYPE_ID Value="STANDARD"/>

<STATE_ID Value="SENT_FOR_FULFILLMENT"/>

<ESTMT_VALUE Value="59554.11"/>

<CURRENCY_CODE Value="USD"/>

<BILL_TO_ID Value="BT_1"/>

<SHIP_TO_ID Value="SP_1"/>

</CUSTOMER_ORDER>

</CUSTOMER_ORDERS>

</RESPONSE>
```

# Loading the Search Definition

The different search definitions that a service has need to be specified in the service config xml file. The search definition files are defined in tag <searchDefnFiles>.

Search definition files can also be reloaded at runtime through specifying the file path at xcore command line and specifying the owner service name.

Service config file:

```
<searchDefnFiles>
<searchDefnFile Name="xservice/orderadmin/xml/sce/search/
coSearchDef.xml"/>
<searchDefnFile Name="xservice/orderadmin/xml/sce/search/
aoSearchDef.xml"/>
<searchDefnFile Name="xservice/orderadmin/xml/sce/search/
```

```
shSearchDef.xml"/>
</searchDefnFiles>
```

# Appendix 1

Example of a search query and its response for the search spec given in example:

```
<REQUESTS ServiceName="ORDER_ADMIN" UserId="user3">

<REQUEST Name="getCustomerOrders" MaxRows="10" ReturnRowCount="yes"
StartAtRow="0" name="//OrderAdminService/getCustomerOrders"
UserId="user3"><FILTER Name="NAME"/>

<FILTER Name="ID">

<ID Value="%" MatchBy="STARTS_WITH"/>

</FILTER><FILTER Name="CREATION_DATE"/>

<FILTER Name="ORDER_POINT_ID"/>

<FILTER Name="ORDER_POINT_NAME"/>

<FILTER Name="ORDER_TYPE_ID">

<ORDER_TYPE_ID Value="STANDARD" MatchBy="EQUAL"/></FILTER>

<FILTER Name="ITEM_ID"><ITEM_ID Value="151-S" MatchBy="EQUAL"/>

</FILTER>

<FILTER Name="STATE_ID">

<STATE_ID Value="INITIATED" MatchBy="NOT_EQUAL"/>

</FILTER><FILTER Name="DOC_TYPE">

<DOC_TYPE Value="ORDER"/>

</FILTER>

<ORDER_BY><CREATION_DATE Sort="Ascending"/>

</ORDER_BY>

</REQUEST>

</REQUESTS>

<RESPONSE Status="Success">

<CUSTOMER_ORDERS TotalRowCount="21" Status="Success"><CUSTOMER_ORDER
ExistingDocument="yes">

<ID Value="CO-1"/><ORDER_POINT_ID Value="OP01"/>

<CREATION_DATE Value="09/22/2003 21:50:28:000"/>

<CREATED_BY_ID Value="user3"/>
```

```
<NAME Value="RESULTSET_UPDATEs"/>
<DOC_TYPE Value="ORDER"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<STATE_ID Value="SENT_FOR_FULFILLMENT"/>
<ESTMT_VALUE Value="59554.11"/>
<CURRENCY_CODE Value="USD"/>
<PRICE Value="9187.743"/>
<ITEM_ID Value="151-S"/>
<REQUEST_DATE Value="09/23/2003 00:00:00:000"/>
<PROMISE_DATE Value="09/23/2003 00:00:00:000"/>
<NOTES><ORDER_NOTES ExistingDocument="yes">
<NOTE_TYPE_ID Value="SHIP_INSTRUCTION"/>
<DOC_ID Value="CO-1"/>
</ORDER_NOTES>
</NOTES>
</CUSTOMER_ORDER>
<CUSTOMER_ORDER ExistingDocument="yes">
<ID Value="CO-103"/>
<ORDER_POINT_ID Value="OP01"/>
<CREATION_DATE Value="09/23/2003 14:33:40:000"/>
<CREATED_BY_ID Value="user3"/>
<NAME Value="RESULTSET_UPDATEs"/>
<DOC_TYPE Value="ORDER"/>
<ORDER_TYPE_ID Value="STANDARD"/>
<STATE_ID Value="SENT_FOR_FULFILLMENT"/>
<ESTMT_VALUE Value="59554.11"/>
<CURRENCY_CODE Value="USD"/>
<PRICE Value="9187.743"/>
<ITEM_ID Value="151-S"/>
<REQUEST_DATE Value="09/23/2003 00:00:00:000"/>
<PROMISE_DATE Value="09/23/2003 00:00:00:000"/>
<NOTES/>
```

```
</CUSTOMER_ORDER>
</CUSTOMER_ORDERS>
</RESPONSE>
```

# APPENDIX Q State Machine

### What's In This Appendix

This chapter gives you information onState Machine Framework.

Topics include:


- Introduction
- State Model
- Mapping Condition Specification and Event Mapping
- State Model Association
- State Document
- Interfaces to State Machine
- Loading State Model Files
- Parallel States
- Example of a State Model Spec


## Introduction

The state machine associates, changes and tracks states associated with document instances. It does this based on a state model specification that is defined for a document instance. A document instance in Order Management parlance is a customer or purchase order, order line item or order schedule etc.

## State Model

A state model defines the set of states a document instance can go through in its life cycle. The set of defined states have an inherent order or rank associated with it. The model also specifies how an instance reaches a particular state. There can be two possible means of reaching a particular state. One is through an event that is raised outside of the state machine and the other is through mapping which is caused as a result of an event within the state machine.

An event is an externally triggered state change. In OMx, the event is triggered during or as a result of a business process. Upon this event, the state machine will change the state of the document instance to the state defined by the event.

This state change may require a state change in a related document instance. The state machine will consider the document instance associations and trigger state change on the related document instance through mapping.

## State Model specification

A state model as in Table 1, specifies the following

1. State model name: This name is unique within a given service
2. Document associated with the state model. The document is defined in xcore as an xdoc.
3. States:
   - An ordered set of states that a document instance goes through.
   - The first state has the lowest rank and the last specified state has the highest rank. These ranks are used in state mapping to decide which state the document instance has to be in when it satisfies more than state mapping conditions.
4. State triggers: The trigger can be an event or a mapping or both, but not more than one of each type.
   - Event trigger: Name of the event that causes the document instance to move to the specified state.
   - Mapping trigger: The condition that has to be satisfied for a document instance to move to a particular state.
5. State model associations or links, if any
   - By default, xdoc document links are used to associate state models.
   - Alternatively, state model links can be defined explicitly in the state model definition.
6. Activation actions:
   - Actions to be executed synchronously when an instance achieves a particular state.
   - Note the syntax of specifying the actions.
7. Events: Events that ought to be raised when a state condition is satisfied

*Table 1: Example of a State Model specification*

```
<state_model Name="MerchandiseLineSM" Document="ORDER_LINE_ITEM">
<state Name="CREATED">
<on_event Name="Created"/>
</state><state Name="DELIVERED">
<on_event Name="Delivered"/>
</state></state_model>
<!-- Name of state model and the associated document -->
<state_model Name="InstallLineSM" Document="ORDER_LINE_ITEM">
<!-- Ordered set of states and the meachanisms through which the
state is reached--><state Name="CREATED">
```

```
<!-- Event 'Created' causes the line item instance to move to CREATED
state.-->

<on_event Name="Created"/>

<!-- The mapping condition that has to be satisfied for the document
instance to get toe CREATED state-->

<on_mapping>

<!-- This mapping condition specifies a state model on which this
state model depends.-->

<link StateModel="MerchandiseLineSM">

<!-- This mapping condition is satisfied when all the linked
MerchandiseLineSM instances have been CREATED-->

<all State="CREATED"/>

</link>

</on_mapping></state>

<state Name="READY_FOR_INSTALL">

<on_mapping>

<link StateModel="MerchandiseLineSM">

<all State="DELIVERED"/>

</link>

</on_mapping>

</state>

<state Name="INSTALL_STARTED">

<on_event Name="InstallStarted"/>

<!-- The actions that are executed when the line item reaches state
INSTALL_STARTED-->

<on_activation>

<actions>

<PRINTLN Value="Executing actions on move to state: {$thisState}"/>
<PRINTLN Value="State changed for instance: {$thisDoc}"/>

<REQUEST Name="StateEventOnMappingListener">

<ID Value="{$thisDoc/ID/@Value}"/>

DOCUMENT_NAME Value="INSTALL_ORDER_LINE_ITEM"/>

<STATE_ID Value="{$thisState}"/>

<USER_ID Value="{$thisUser}"/>
```

```
<REASON_CODE Select="$thisParam/REASON_CODE/@Value"/><REASON_TEXT
Select="$thisParam/REASON_TEXT/@Value"/>

</REQUEST>

</actions>

</on_activation>

</state>

<state Name="INSTALL_COMPLETED">

<on_event Name="InstallCompleted"/>

</state>

</state_model>
```

# Mapping Condition Specification and Event Mapping

Mapping conditions can involve one or more linked state models. The logical operations supported are 'and' and 'or'. For a given linked state model, conditions can be specified on the states of document instances of that state model. Currently, the 2 supported operations are 'any' and 'all' with one of the many operators - EQUAL, GREATER_EQUAL, GREATER, LESS, LESS_EQUAL and NOT_EQUAL.

Following is an example illustrating these and how an event can be raised as a result of a mapping condition.

Example of a mapping condition resulting in an event:

```
<event Name="StateEventOnMapping">

<on_mapping>

<or>

<link StateModel="MerchandiseLineSM">

<any State="DELIVERED" MatchBy="EQUAL"/>

<all State="DELIVERED" MatchBy="GREATER_EQUAL"/>

</link>

<link StateModel="InstallLineSM">

<any State="READY_FOR_INSTALL" MatchBy="GREATER_EQUAL"/>

<or>

<all State="INSTALL_STARTED" MatchBy="LESS_EQUAL"/>

<all State="INSTALL_COMPLETED" MatchBy="LESS"/>
```

```
</or>
</link>
</or>
</on_mapping>
</event>
```

# State Model Association

The state machine figures out the linked state model document instances through state model links. The following table illustrates how to specify links between state models. If no such link is explicitly specified between state models, direct document links as defined in xcore are used. If no defined document link exists between 2 document instances, then an exception is thrown at bootstrap time.

In summary, the state machine supports three types of state model links:

1. Direct document link: Default link and does not have to be explicitly specified as a state model link.
2. Expression link: An Xpath expression is used to associate state models.
3. Multi-path link: A multi-path document link as defined in x-doc is specified to associate state models. The parts are the document link names from x-doc of one state model all the way to the final document link name relating x-doc of the other state model.

State model associations: Expression links

```
<state_model_links>
<!-- This link expression takes as parameter a Merchandise line
instance and returns all linked Install line instances-->
<model_link From="MerchandiseLineSM" To="InstallLineSM"
Value="{request('getLinkedLine', $thisDoc)}"/>
<!-- This link expression takes as parameter an Install line instance
and returns all linked merchandise  line instances-->
<model_link From="InstallLineSM" To="MerchandiseLineSM"
Value="{request('getLinkedLine', $thisDoc)}"/>
</state_model_links>
```

State model associations:

```
Multi-path links <state_model_links>
<model_link From="StandardShipmentLineSM"
To="StandardOrderScheduleSM" Path="SHIP_LN_TO_SCH_LINK/
SCHEDULE_LINK"/>
```

```
<model_link From="StandardOrderScheduleSM"
To="StandardShipmentLineSM" Path="SHIP_LN_TO_SCH_LINK/SHIP_LN_LINK"/
>

</state_model_links>
```

## Exception States

Exception states are those states that will not participate in a mapping condition unless the defined comparison state is also an exception state.

For example, CANCELLED and REJECTED are two exception states of an order line item. These states will be discounted in mapping computation of an order having n number of lines with x (where x < n) lines in CANCELLED state.

An exception state is defined through an attribute of the state definition tag.

Exception states and mapping computation:

```
<state_model>

<!--DELIVERED is a non-exception state -->

<state Name="DELIVERED">

<!-- All linked exception states will be ignored in this mapping
evaluation --><on_mapping><link StateModel="StandardLineSM">

<all State="DELIVERED" MatchBy="GREATER_EQUAL"/>

</link>

</on_mapping></state>

<!--CANCELLED is marked as an exception state -->

<state Name="CANCELLED" ExceptionState="true">

<on_mapping>

<!-- MerchandiseLineSM  will have defined CANCELLED as an exception
state.Since this comparison state is an exception state, the mapping
computation will include CANCELLED and REJECTED states -->

<link StateModel="MerchandiseLineSM">

<all State="CANCELLED" MatchBy="GREATER_EQUAL"/></link></on_mapping>

</state>

<state Name="REJECTED" ExceptionState="true">

<on_mapping><link StateModel="MerchandiseLineSM">

<all State="REJECTED" MatchBy="EQUAL"/>

</link>

</on_mapping>
```

```
</state>

</state_model>
```

# State Document

A state document is an extended characterization of the xcore document object maintained by the State Machine for its working. Any document in xdoc definition can be used in state machine. If a user wants to associate an x-doc with state models, then the user has to add 3 document properties. These are: STATE_MODEL, STATE_ID and STATE_HISTORY.

During state model specification, these three properties are considered default by the state machine and do not have to be specified explicitly as the state properties for the state model.

If the user wishes to use different property names in x-doc to store the state related data, then these properties can be described in the state model. Refer to section on Parallel States.

The STATE_MODEL property value is specified by the application using the state machine. The other two properties are maintained by the state machine. STATE_ID defines the current state of the document instance while STATE_HISTORY is a concatenated string of all the previous states.

It is good practice to define an index constituting the primary keys and the State model property in that order. This is because the state machine accesses document instances using the primary keys and the state model.

The user may choose to specify additional state properties in the x-doc definition if document has a notion of parallel states.

For purposes of backward compatibility, DOCUMENT defined as STATE_DOCUMENT in x-doc definition is still supported through specifying document handler com.teradata.xcore.tool.DocumentHandler for STATE_DOCUMENT.

Setting the State Model:

Applications can use xrule action tag SET_STATE_MODEL to set the state model for a given document instance.

Example of SET_STATE_MODEL:

```
<SET_STATE_MODEL Select="$thisParam/ORDER_LINE_ITEM"
FromValue="{$sm}"/>OR <SET_STATE_MODEL Select="$thisParam/
ORDER_LINE_ITEM" FromValue="MerchandiseLineSM"/>
```

# Interfaces to State Machine

There are two primary interfaces to the state machine. One of them is to change the state of a list of document instances through a state change event and the other is to associate the immediate previous state of a document instance as the current state.

Other interfaces include, obtaining the event name that causes a document instance to reach a particular state.

## RAISE_STATE_EVENT

This is the action tag to be used to change the state of a list of entities given an event name. The event name has to be valid for the all state models of the parameter instances.

The instances themselves can have different state models.

By default, the state machine will query the database to obtain a new copy of the instance data, so that all properties of each document instance are available for the state machine to work with. This query will use the parameters passed to RAISE_STATE_EVENT as the context.

If users are sure that they are passing all the relevant data, they can specify an attribute `Refetch="false"`, to prevent the state machine from making this database query.

If the state of the parameter document instances change, then the states of all mapped document instances will be reevaluated.

Example of RAISE_STATE_EVENT:

```
<!-- The expression for Name has to evaluate to a valid event name
for the state models specified in each of the parameter instances -->

<RAISE_STATE_EVENT Name="{$thisParam/EVENT/@Value}" Refetch="false">
<ORDER_LINE_ITEM>

<ID Value="LI-190"/>

<ORDER_TYPE_ID Value="MERCHANDISE"/>

<ORDERED_QTY Value="3"/>

<STATE_MODEL Value="MerchandiseLineSM"/></ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>

<ID Value="LI-819"/>

<ORDER_TYPE_ID Value="MERCHANDISE"/>

<ORDERED_QTY Value="4"/>

<STATE_MODEL Value="MerchandiseLineSM"/></ORDER_LINE_ITEM>

<ORDER_LINE_ITEM><ID Value="LI-900"/><OR

DER_TYPE_ID Value="INSTALL"/><ORDERED_QTY Value="1"/><STATE_MODEL
Value="InstallLineSM"/></ORDER_LINE_ITEM>

</RAISE_STATE_EVENT>
```

## PREVIOUS_STATE_EVENT

This is the action tag to be used to change the state of all the parameter instance documents to their previous state. After the states of these instances are changed to their previous state, all mappings that these instances participate in will be reevaluated.

If no previous state exists for any instance, then an exception will be thrown. Any mapping that is triggered as a result of this state change, will throw an exception if no valid state is detected.

Example of PREVIOUS_STATE_EVENT:

```
<PREVIOUS_STATE_EVENT Refetch="false">
<ORDER_LINE_ITEM>
<ID Value="LI-190"/>
<ORDER_TYPE_ID Value="MERCHANDISE"/>
<ORDERED_QTY Value="3"/>
<STATE_MODEL Value="MerchandiseLineSM"/>
</ORDER_LINE_ITEM>
<ORDER_LINE_ITEM>
<ID Value="LI-819"/>
<ORDER_TYPE_ID Value="MERCHANDISE"/>
<ORDERED_QTY Value="4"/>
<STATE_MODEL Value="MerchandiseLineSM"/>
</ORDER_LINE_ITEM><ORDER_LINE_ITEM>
<ID Value="LI-900"/>
<ORDER_TYPE_ID Value="INSTALL"/>
<ORDERED_QTY Value="1"/>
<STATE_MODEL Value="InstallLineSM"/>
</ORDER_LINE_ITEM>
</PREVIOUS_STATE_EVENT>
```

## Obtaining event name for a state in a state model

This function will return the event that is defined to trigger a state for a given state model.

If the state or state model is invalid, an exception is thrown. If they are valid and no event exists for the state, null is returned.

```
Function: getEventState
<SET Var="event" FromValue="{getStateEvent('StateModelName',
'StateName')}"/>
```

OR

```
<SET Var="event" FromValue="{getStateEvent($thisParam/MODEL/@Value,
$thisParam/STATE/@Value)}"/>
```

# Loading State Model Files

State models file names are specified in the service configuration file of the service they are to be registered with. All calls to the state machine involving these state models will have to occur within that service.

Handler for state model files:

```
<register-handlers>

<handler TagName="stateModelFiles"
ClassName="com.teradata.xcore.statemachine.StateModelFilesHandler"/>

</register-handlers>
```

Registration of state model:

```
<stateModelFiles>

<stateModelFile Name="xservice/orderadmin/StateModel- SellSide.xml"/
>

</stateModelFiles>
```

# Parallel States

The state machine can track and support multiple state models on the same x-doc. In order to do this:

- x-doc definition must contain all the state properties used in its document definition.
- Each state model specification excepting the one using default state properties (STATE_ID, STATE_MODEL, STATE_HISTORY) must specify the state properties to be used to maintain the states associated with that state model.
- If a state event has to be raised on a document instance for a state model other than the default, then a StateModel attribute must be mentioned while raising the state event.
- State model links must be specified as done in the default one.

The state machine will then use the state properties associated with the state model to track and maintain the parallel states.

*Example 1:*

A state model defining and using a specific set of state properties.

```
<state_models>
```

```
<state_model Name="pceScheduleInvSM" Document="ORDER_SCHEDULE">
<state_properties>
<state_id Name="INVOICE_STATUS"/>
<state_model Name="INV_STATUS_SM"/>
<state_history Name="INV_STATUS_HISTORY"/>
</state_properties>
<state Name="NOT_ELIGIBLE">
<on_event Name="NotEligible"/>
</state>
<state Name="PENDING">
<on_mapping>
<link StateModel="StandardBuyOrderSchSM">
<all State="ACCEPTED" MatchBy="GREATER_EQUAL"/>
<all State="PARTIALLY_INVOICED" MatchBy="LESS"/>
</link>
</on_mapping>
</state>
<state Name="PARTIALLY_INVOICED">
<on_mapping>
<link StateModel="StandardBuyOrderSchSM">
<all State="PARTIALLY_INVOICED" MatchBy="EQUAL"/>
</link>
</on_mapping>
</state>
<state Name="INVOICED">
<on_mapping>
<link StateModel="StandardBuyOrderSchSM">
<all State="INVOICED" MatchBy="EQUAL"/>
</link>
</on_mapping>
</state>
</state_model>
</state_models>
```

*Example 2:*

Raising state event on a non-default state model. Not the StateModel attribute that is specified. The same can be specified for PREVIOUS_STATE_EVENT as well.

```
<RAISE_STATE_EVENT Name="Invoiced" StateModel="ScheduleInvoiceSM">
<ORDER_SCHEDULE>
<ID Value="SCH-190"/>
</ORDER_SCHEDULE>
</RAISE_STATE_EVENT>
```

# Example of a State Model Spec

A combination customer order's states are determined by its constituent merchandise and install lines. The install line states are in turn determined by the merchandise line states.This example also shows state model links as defined in the state model.

```
<state_models>
<state_model Name="CombinationOrderSM" Document="CUSTOMER_ORDER">
<on_activation_any_state>
<actions>
<PRINTLN Value="Executing actions on move to state: {$thisState}"/>
<PRINTLN Value="State changed for instance: {$thisDoc}"/>
</actions>
</on_activation_any_state>
<state Name="CREATED">
<on_activation>
<actions>
<PRINTLN Value="Customer order is created: {$thisDoc/ID/@Value}"/>
</actions>
</on_activation>
<on_event Name="Created"/>
<on_mapping>
<link StateModel="MerchandiseLineSM">
<all State="CREATED"/>
</link>
<link StateModel="InstallLineSM">
```

```
<all State="CREATED"/>
</link>
</on_mapping>
</state>
<event Name="StateEventOnMapping">
<on_mapping>
<or>
<link StateModel="MerchandiseLineSM">
<any State="DELIVERED" MatchBy="EQUAL"/>
</link>
<link StateModel="InstallLineSM">
<any State="READY_FOR_INSTALL" MatchBy="GREATER_EQUAL"/>
</link>
</or>
</on_mapping>
</event>
<state Name="MERCHANDISE_PARTIALLY_DELIVERED">
<on_mapping>
<link StateModel="MerchandiseLineSM">
<any State="DELIVERED"/>
</link>
</on_mapping>
</state>
<state Name="MERCHANDISE_DELIVERED">
<on_mapping><link Doc="ORDER_LINE_ITEM"
StateModel="MerchandiseLineSM">
<all State="DELIVERED"/>
</link>
</on_mapping>
</state>
<state Name="INSTALL_IN_PROGRESS">
<on_mapping>
<link Doc="ORDER_LINE_ITEM" StateModel="InstallLineSM">
```

```
<any State="INSTALL_STARTED"/>

</link>

</on_mapping>

</state><state Name="COMPLETED">

<on_mapping>

<link Doc="ORDER_LINE_ITEM" StateModel="MerchandiseLineSM"><all
State="DELIVERED"/>

</link>

<link Doc="ORDER_LINE_ITEM" StateModel="InstallLineSM">

<any State="INSTALL_COMPLETED"/>

</link>

</on_mapping>

</state>

<state Name="CANCELLED" ExceptionState="true">

<on_event Name="Cancelled"/>

</state>

</state_model>

<state_model Name="MerchandiseLineSM" Document="ORDER_LINE_ITEM">
<state Name="CREATED">

<on_event Name="Created"/>

</state><state Name="DELIVERED">

<on_event Name="Delivered"/>

</state>

</state_model>

<!-- Name of state model and the associated document -->

<state_model Name="InstallLineSM" Document="ORDER_LINE_ITEM">

<!-- Ordered set of states and the meachanisms through which the
state is reached--><state Name="CREATED">

<!-- Event 'Created' causes the line item instance to move to CREATED
state.--><on_event Name="Created"/>

<!-- The mapping condition that has to be satisfied for the document
instance to get toe CREATED state-->

<on_mapping>
```

```
<!-- This mapping condition specifies a state model on which this
state model depends.--><link StateModel="MerchandiseLineSM">

<!-- This mapping condition is satisfied when all the linked
MerchandiseLineSM instances have been CREATED-->

<all State="CREATED"/>

</link>

</on_mapping>

</state>

<state Name="READY_FOR_INSTALL">

<on_mapping><link StateModel="MerchandiseLineSM">

<all State="DELIVERED"/>

</link>

</on_mapping>

</state>

<state Name="INSTALL_STARTED">

<on_event Name="InstallStarted"/>

<on_activation>

<actions>

<PRINTLN Value="Executing actions on move to state: {$thisState}"/>

<PRINTLN Value="State changed for instance: {$thisDoc}"/><REQUEST
Name="StateEventOnMappingListener">

<ID Value="{$thisDoc/ID/@Value}"/>

<DOCUMENT_NAME Value="INSTALL_ORDER_LINE_ITEM"/><STATE_ID
Value="{$thisState}"/>

<USER_ID Value="{$thisUser}"/>

<REASON_CODE Select="$thisParam/REASON_CODE/@Value"/>

<REASON_TEXT Select="$thisParam/REASON_TEXT/@Value"/></REQUEST></
actions>

</on_activation>

</state>

<state Name="INSTALL_COMPLETED">

<on_event Name="InstallCompleted"/>

</state>

</state_model>
```

```
<state_model_links>

<!-- This link expression takes as parameter a Merchandise line
instance and returns all linked Install line instances-->

<model_link From="MerchandiseLineSM" To="InstallLineSM"
Value="{request('getLinkedLine', $thisDoc)}"/>

<!-- This link expression takes as parameter an Install line instance
and returns all linked merchandise  line instances-->

<model_link From="InstallLineSM" To="MerchandiseLineSM"
Value="{request('getLinkedLine', $thisDoc)}"/>

</state_model_links></state_models>
```

# FAQ

**Q.** Does a state model have to be associated with a state document?

**A.** Yes, a state model has to be associated with a state document.

**Q.** Can multiple state models be defined for the same state document?

**A.** Yes, multiple state models with different names can be associated with the same state document. Each document instance of that state document will use one state model.

**Q.** If a state in the state model is defined as triggered by both an event and mapping, which one takes precedence?

**A.** If the state can be reached through either one of the mechanisms, the document instance changes to that state if its current state is less than the new state. The mechanism does not matter in state transition.

**Q.** How does transition to a lower state occur during state mapping?

**A.** If a condition defined at a higher state than current state is satisfied, then transition to the higher state will occur. When no condition for state higher than current state is satisfied, then the last valid state of the document instance is computed. The last valid state is the highest state in the state history of the instance that has an event trigger or has a mapping condition that holds true. Even if this state was not reached using the event, this state is still deemed to be valid.

**Q.** What is the outcome of an event mapping?

**A.** If the mapping condition associated with an event evaluates to true, then the event is raised with the document instance as the parameter. This event is by definition asynchronous. Hence, the event will have to defined and registered in the event document for the service.

**Q.** Is the interface for RAISE_STATE_EVENT backward compatible with previous definitions of RAISE_STATE_EVENT?

**A.** Yes, it is. One can use FOR_DOCUMENT and DOCUMENT_CONTEXT tags. However, these will be deprecated. The preferred way is to simply list the document instances under the RAISE_STATE_EVENT tag.

**Q.** When can I safely set the Refetch attribute of RAISE_STATE_EVENT or PREVIOUS_STATE_EVENT to false?

**A.** The state machine event and mappings work on the document instance. If you are sure that the instance passed as parameter is current and contains all the information that the state machine will require for evaluating mappings (including document link properties used in fetching linked instances), then set Refetch to false.

# APPENDIX R Workflow Specification Framework

### What's In This Appendix

This appendix provides information on the workflow specification and its framework.

Topics include:


- Introduction
- Built-In Workflow Nodes


# Introduction

The workflow diagram mimics UML's Activity diagram where the activities are executed within nodes.

Workflow engine execute the workflow through its entirety, some code initiates the workflow, the initiation of the workflow and interfaces of workflow engine are not part of this specification.

Every workflow has the following:

- Name.
- A set of nodes
- Set of predefined variables called workflow variables. These workflow variables are global to the workflow. They can be mutated by actions in various nodes of the workflow. The variable types are limited to the data-types natively supported by X-Core: string, int, float, double, date, datetime, time, Boolean and Xml. Please refer Xcore programmers manual for more details on X-Core data types.

You can specify only one workflow in a file. Here is an example of a workflow:

## Typical Workflow Example

```
<workflow Name="subWorkflowTest">

<variables>

<variable Name="orderId" Type="string(32)"/>

</variables>

<nodes>
```

```
Set of workflow nodes

</nodes>

</workflow>
```

A workflow node can optionally have an input and output. The input can be accessed in the workflow actions by referring to the input Var name.

The output variable can be a workflow variable or any other local variable created in the action statements.

## Typical Node Example

```
<nodetype Name="some-name" InputVar="{VarName, VarType}" >

<actions><action Name="some action name1">

.......... X-Rule action statements ...................
.....................................................................
.

</action>

<action Name="some action name2">

.......... X-Rule action statements ...................
.....................................................................
.

</action></actions>
<next_nodes OutputVar="{VarName, VarType}">

<next_node Name="some-node-name1"/>

<next_node Name="some-node-name2"/>

</next_nodes>

</nodetype>
```

All of the workflow nodes have access to the implicit return variable, thisReturn.

## Transaction Scope in Workflow

A workflow is kicked off through a start node and it continues execution until it reaches a quiescent state. A quiescent state is one where all the workflow paths reach a event, timer, sub-workflow node or done node. Once the workflow reaches done node through any path, then it ceases to exist. The workflow resumes execution when one of the nodes, on which the workflow is waiting, is triggered. Again the workflow continues execution until it reaches a quiescent state. The workflow execution, until it reaches a quiescent state, is performed in a single transaction. The transaction is same as the thread that initiated the workflow by explicit instantiation or by triggering a event.

## Event Descriptors

X-Core Events and Requests: Events can be raised in X-Core through <RAISE_EVENT> command. There can be multiple listeners for an event. The thread raising the event will not get any of the responses spewed out by the listeners. Requests are invoked through <REQUEST> command. A request is mapped to a single handler. The response of the handler is returned to the invoker of the request.

A workflow event node can wait for external/internal Requests or internally generated events. Only one workflow event node can wait for a request with the same key values at any given time. However you can have several workflow event nodes waiting at the same time on an event with the same key values.

An event/request is identified in a event descriptor file by:

- Name
- Keys. An event can have multiple keys and each key value is specified through an X-Path expression.
- A flag (IsRequest) identifying if it is a request or an event. The default for this flag is false.
- You can optionally specify a DefaultHandler attribute for Request. The default handler is a rule that will be called if there are no event nodes with matching keys waiting for that Request. The default handler is especially useful for formatting error response to the invoker.

Example of an event descriptor file:

```
<event_descriptors>

<event_descriptor Name="ACTIVITY_ORDER.State.CANCEL.Activated"><key
Value="$thisParam/ACTIVITY_ORDER/ID/@Value"/>

</event_descriptor>

<event_descriptor Name="acknowledgeActivityOrder" IsRequest="true"
DefaultHandler="illegalAckEventHandler">

<key Value="$thisParam/PO_ACK/AO_ID/@Value"/>

</event_descriptor>

<event_descriptor Name="cancelActivityOrder" IsRequest ="true"><key
Value="$thisParam/ACTIVITY_ORDER/ID/@Value"/>

</event_descriptor>

</event_descriptors>
```

The workflow engine matches the event nodes of the workflow instances to the corresponding request/event by matching the request/event name and keys with the event descriptor and matching keys of the event node respectively.

# Built-In Workflow Nodes

There are several available nodetypes:

- Start Node
- Task Node
- Decision Node
- Branch Node
- Event Node
- And Node
- Or Node
- Timer Node
- Sub-Workflow Node
- Done Node

All nodes have grammar similar to the nodetype grammar shown above with some variations. Below are description of these nodes and their variations.

## Start Node

The grammar is same as nodetype where nodetype is start. As the name implies, this is the starting node for the workflow.

There cannot be any nodes referring to the start node. The start node can refer to the other nodes.

A workflow can be started in two ways:

1 By Explicit instantiation using START_WORKFLOW Command.

The workflow kicks off starting from the default start node. There can be only one default start node in a workflow.

Any start node without an event descriptor is treated as a default start node.

Here is an example of explicit instantiation of a workflow:

<START_WORKFLOW >

<TEMPLATE Value="orderCreateWfl" />

<ADDITIONAL_PARAMETERS>

<ORDER_ID Value="ddd" />

<MERCHANT_ID Value="mmm" />

</ADDITIONAL_PARAMETERS>

</START_WORKFLOW>

The template is the name of the workflow.

*Example:*

Start Node triggered only by explicit instantiation:

<start Name="start">

<action>

… some glue code …

</action>

<next_nodes>

<next_node Name="DecideAfterChangeCreate"/>

</next_nodes>

</start>

This node will also be the default node since it is not associated with any event descriptor

2 By an event.

A workflow can be started by an event. You can raise events in X-Core using the RAISE_EVENT command

Here is an example:

```
<RAISE_EVENT Name="CUSTOMER_ORDER.State.SOLD.Activation">
<CUSTOMER_ORDER>
<ID Value="abc" />
<EXT_ORDER_NUM Value="x12345" />
.............................
</CUSTOMER_ORDER>
</RAISE_EVENT>
```

Providing an event descriptor (by specifying the Descriptor attribute) in the start node will enable the workflow to be triggered by that event. You can have only one start node with the same event descriptor. Please note that you cannot provide any matching keys for the event descriptor in the start node. So the start node behaves like a global function.

*Example:*

Start Node triggered by an event:

```
<start Name="start2" Descriptor="receiveShipment"
IsDefault="false">
<action>
… some glue code …
</action>
<next_nodes>
<next_node Name="DecideAfterChangeCreate"/>
</next_nodes>
</start>
```

**Note:** `receiveShipment` should be one of the events defined in the event descriptor file

You can also flag a start node with a event descriptor as a default node. This start node can then be triggered by explicit instantiation or by an event.

*Example:*

Start Node triggered by an event or by explicit instantiation

```
<start Name="start2" Descriptor="receiveShipment"
IsDefault="true">
<action>
… some glue code …
</action>
<next_nodes>
<next_node Name="DecideAfterChangeCreate"/>
</next_nodes>
</start>
```

**Note:** `receiveShipment` should be one of the events defined in the event descriptor file.

## Task Node

The grammar is same as nodetype where nodetype is task. The node's intent is to carry out some activity.

An example is as below:

Task Node:

```
<task Name="transmit PO" Description="Transmit to PO Routing Service"
InputVar="{aoDoc, Xml}">
<actions>
<action Name="transmit the PO">
<REQUEST ServiceName="INTEGRATION" Name="TransmitPo">
<TO_XML DocVar="aoDoc"/>
</REQUEST>
</action>
</actions>
<next_nodes  OutputVar="{aoDoc, Xml}">
<next_node  Name="isPendingCancel"/>
</next_nodes>
</task>
```

A task node can optionally receive some input

## Decision Node

The grammar is almost same as nodetype. The nodetype is decision and hence it has a decision condition. The decision condition is an Xpath expression. Based on the decision condition, you can either go to if_true or if_false next nodes. There is no next_nodes for the decision node.

The example is below:

```
<decision Name="isRejectionAllowed" Description="Check if the AO is
rejected or accepted">
<actions>
<action Name="Get AO Type">
<SET Var="aoType" FromSelect="$aoDoc/AO_TYPE/@Value"/>
</action>
</actions>
<condition Value="$aoType='SPECIAL'"/>
<if_true>
<next_nodes OutputVar="{ aoDoc, Xml}">
<next_node Name="rejectLineItems"/>
</next_nodes>
</if_true>
<if_false>
<next_nodes OutputVar="{ aoDoc, Xml}">
<next_node Name="throwRejectException"/>
</next_nodes>
</if_false>
</decision>
```

## Branch Node

The grammar is almost same as nodetype. The nodetype is branch and there is no <next_nodes> tag. It has if and else-if conditions where the conditions are specified by X-Path expressions.

The example is below:

```
<branch  Name="switchWfOnType" Description="start Wf on type"
InputVar="{aoDoc,Xml}">
<actions>
<action Name="Get AO Type">
<SET Var="aoType" FromSelect="$aoDoc/AO_TYPE/@Value"/>
</action>
</actions>
<if_cond Value="$aoType='MEASURE'" >
<next_nodes OutputVar="{aoDoc,Xml}">
```

```
<next_node Name="measureSubWf"/>
</next_nodes></if_cond>
<else_if_cond Value="$aoType='INSTALL'" >
<next_nodes OutputVar="{aoDoc,Xml}">
<next_node Name="installSubWf"/>
</next_nodes>
</else_if_cond>
<else_if_cond Value="$aoType='DELIVERY'" >
<next_nodes OutputVar="{aoDoc,Xml}">
<next_node Name="deliverySubWf"/>
</next_nodes>
</else_if_cond>
<else_if_cond Value="$aoType='SPECIAL'">
<next_nodes OutputVar="{aoDoc,Xml}">
<next_node Name="switchWfOnShipTo"/>
</next_nodes>
</else_if_cond>
</branch>
```

## Event Node

The grammar is same as nodetype. It also has an attribute named "Descriptor" which specifies the name of the event. The descriptor must be one of the event descriptors specified in event descriptor file. There is an extra tag named `<keys>` that specifies the matching keys for the event. The matching key values are specified as X-Path expressions. The number of keys must match those defined in the event descriptor file for that event. The Workflow engine will halt the execution of workflow until the specified event with the matching keys is raised. The actions are executed after the event is triggered.

The example is below:

```
<event Name="ChangePromiseEventListener"
Descriptor="changePromiseEvent"><keys>
<key Value="$coId"/>
</keys>
<actions>
… some glue code …
</actions>
```

```
<next_nodes>
<next_node Name="DecideAfterChangePromise"/>
</next_nodes>
</event>
```

## And Node

The grammar is same as nodetype where nodetype is and_join. When the node is reached through all the input paths, the activity is executed.

The example is below:

<and_join Name="And Node">

<actions>

… some glue code …

</actions>

<next_nodes>

<next_node Name="Branch Node"/>

</next_nodes>

</and_join>

## Or Node

The grammar is same as nodetype where nodetype is or_join. When the node is reached through atleast one input path then the activity is executed.

The example is below:

```
<or_join Name="Or Node">
<actions>
<action Name="testingor">
<PRINT Value="testingor"/>
</action>
</actions>
<next_nodes>
<next_node Name="End Node"/>
</next_nodes>
</or_join>
```

## Timer Node

The workflow will wait on this node until the date-time specified by the expiration date is reached. You can perform:

- pre-actions before waiting for the expiration date
- post-actions after the expiration date is reached.

The expiration date can be specified by X-Path expression. The expiration date will be computed after executing the pre-actions. After the timer node is done, you can move on to the next nodes.

The example is below:

```
<timer Name="startPromisedShipDateTimer" Description="Start Timer
for Promised Ship Date" InputVar="{aoDoc,Xml}">

<expiration_date Value="date($liDoc/PROMISED_SHIP_DATE/@Value)"/>
<pre_actions>

<action Name="Get the first active line item">

<SET Var="liDoc" FromValue="{$aoDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM[STATE_ID!='CANCEL']}"/>

</action>

</pre_actions>

<post_actions>

<action Name="Get Ao Doc">

<SET Var="aoDoc" FromSelect="request('getAoDoc','ID', $aoId)/
ACTIVITY_ORDER"/>

</action>

<action Name="Set all  Li States to Shipped">

<FOR_EACH SelectList="$aoDoc/ORDER_LINE_ITEMS/
ORDER_LINE_ITEM[STATE_ID!='CANCEL']" CurrentElement="liDoc">

<RAISE_STATE_EVENT Entity="liDoc" Name="Shipped"/></FOR_EACH></
action>

</post_actions><next_nodes OutputVar="{aoDoc,Xml}"><next_node
Name="costSubWf"/>

</next_nodes>

</timer>
```

## Sub-Workflow Node

This node is for starting sub-workflows. The workflow will wait on this node until the sub-workflow is done. You can perform:

- pre-actions before starting the sub workflow
- post-actions after the sub-workflow is done.

You can specify the input to the sub-workflow using X-Path expressions. The input to the sub-workflow will be evaluated after executing the pre actions. You can also assign the output of the sub-workflow to a variable and use them in your post actions. After the sub workflow node is done, you can move on to the next nodes.

The example is below:

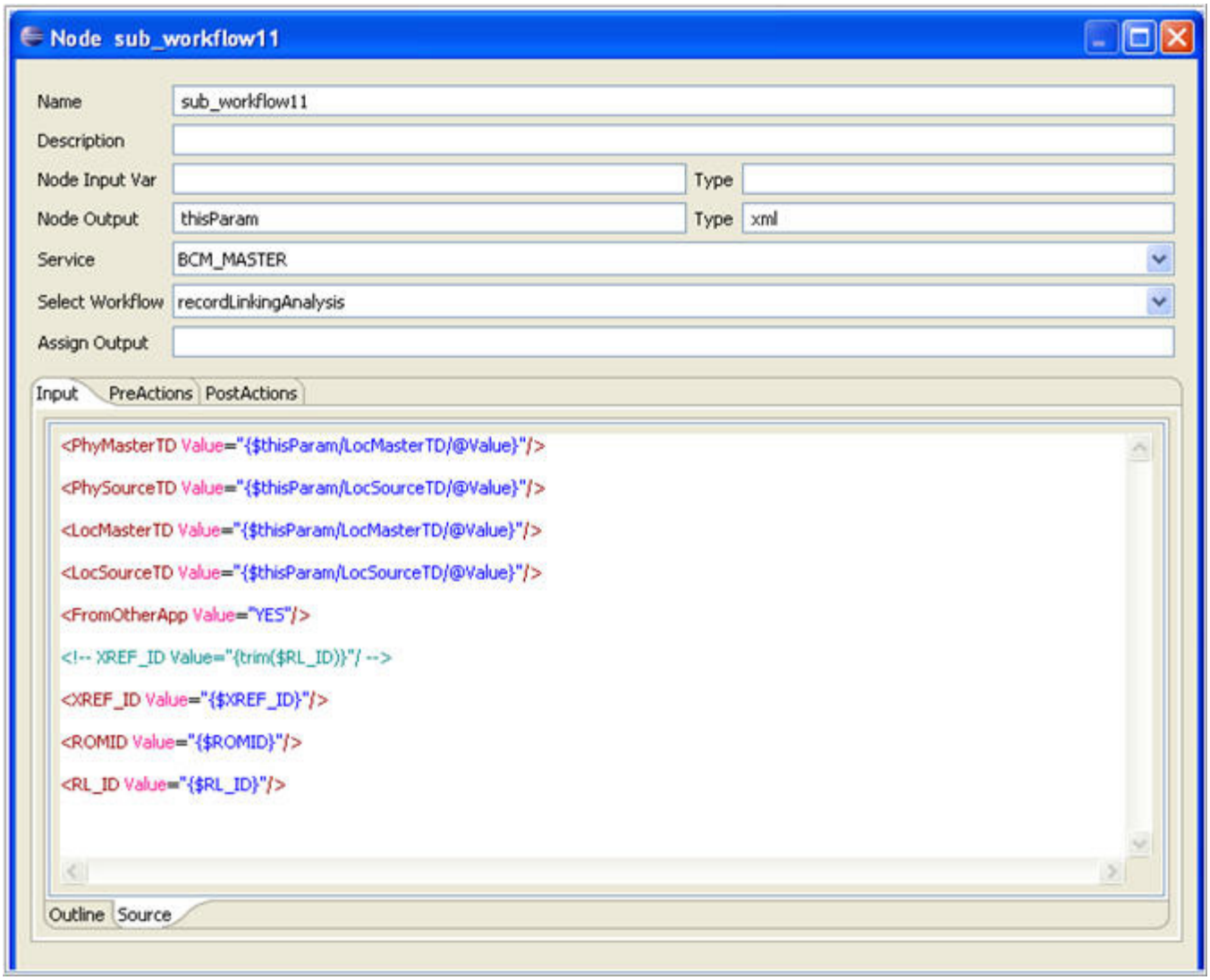
```
<sub_workflow Name="costSubWf" Workflow="aoCostWf"
InputVar="{aoDoc,Xml}">

<input>

<TO_XML DocVar="aoDoc"/>

</input>

<output AssignToVar="subWflReturn"/>

<pre_actions>

<action Name="Print Ao Doc">

<PRINTLN Value="{$aoDoc}" />

</action>

</pre_actions>

<post_actions>

<action Name="Print Sub Wfl Return">

<PRINTLN Value="{$subWflReturn}" />

</action>

<action Name="get AO Doc">

<SET Var="aoDoc" FromValue="{request('getADoc','ID', $aoId)/
ACTIVITY_ORDER}"/>

</action>

</post_actions>

<next_nodes OutputVar="{aoDoc,Xml}">

<next_node Name="keyRecSubWf"/>

</next_nodes>

</sub_workflow>
```

**Note:** Sub Workflow doesn't send Input variable to target workflow if define parameter is with in action tag.

For the sub workflow to pass the value to the target value, the workaround for this is given below:

On the sub workflow node editor, click **Input** tap and select **Source** tab, enter the inputs as in Figure 105. In this section, it should not contains the action tag.

Figure 105: Sub Workflow Node




## Done Node

The grammar is almost same as nodetype where nodetype is done. But there are no `<next_nodes>` tag. More then one done nodes are allowed for a workflow.

The example is below:

```
<done Name="done1">
<actions>
<action Name="Do nothing"/>
</actions>
</done>
```

Workflow engines cares for the above grammar and executes the workflow. However all the extraneous attributes and element are ignored. This fact is used to shove in the view adornments.

# APPENDIX S Messaging and E-mail Service

### What's In This Appendix

This appendix provides information on the messaging service and e-mail service of MDM.

Topics include:


- Introduction
- Messaging Functionality
- E-mail Service


## Introduction

This document lays out the detailed description of the Messaging Service and e-mail service t of the MDM application.

## Messaging Functionality

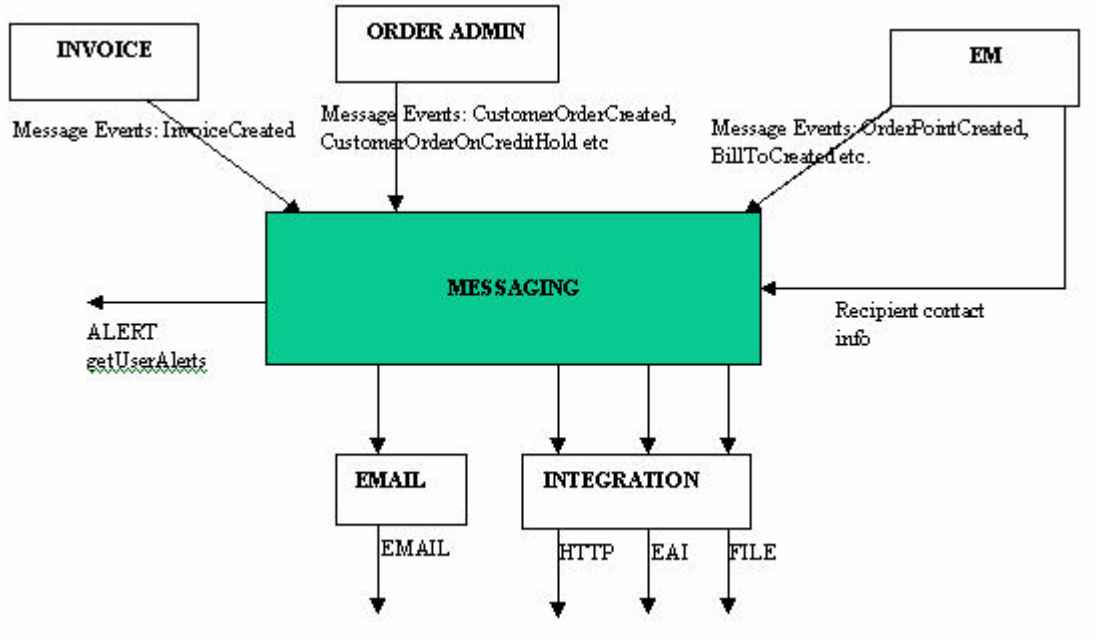
The messaging service is a utility service that can be used by all other services (call them client services) to send messages to different recipients. The client service does not need to worry about all the messaging details like whom to send, how to send, what to send etc. All such settings are configured in the messaging service through xml specification files or through UI. To send a message the client service will raise (send) a predefined message event to the messaging service and then the messaging service will use the event definition to send out required messages.

The messaging service can send messages in five formats: ALERT, EMAIL, HTTP, FILE, EAI (or EDI). For actual transmission in HTTP, FILE and EAI formats the Integration service is used, for EMAIL format the Email service is used and the ALERT format is handled locally.

The messaging service uses the EM service to find out the contact details like email address, output file path etc. The EM service is also used in case of ROLE based recipients to find recipients in a particular ROLE.

The following diagram shows the interactions between messaging service and other services:

Figure 106: Interactions between Messaging service and other services




For a client service to use the message service the steps to be followed are as follows:

1. The client service has to register with the messaging service the message events that it will generate.
2. Provide definition for the actions to perform when the event is raised. The action type supported today is SEND_MESSAGE
3. For the SEND_MESSAGE action define the message template to be used. The message template defines the contents of the message in different protocols (EMAIL, ALERT, HTTP etc).
4. For the SEND_MESSAGE action define the RecipientGroups. RecipientGroup is a logical grouping of recipients. The RecipientGroup can contain one or more recipient. Recipient is a logical entity that has contact information. So the recipient can be a user, an organization, a URL etc.

All these settings are done in xml specification file and the files are uploaded to the messaging service. This file needs to be uploaded only once as the messaging service will read the file and populate internally database tables. If the files are uploaded again the old records in the database will be overwritten.

The command to upload the specification files looks like:

```
<REQUESTS ServiceName="MESSAGING">
    <REQUEST Name="uploadMessagingSpec">
        <FILE_NAME Value="xservice/orderadmin/messaging/
        oaMsgEvents.xml"/>
```

```
        <FILE_NAME Value="xservice/orderadmin/messaging/
        oaMsgEventDefs.xml"/>
        <FILE_NAME Value="xservice/orderadmin/messaging/
        oaMsgTemplates.xml"/>
        <FILE_NAME Value="xservice/orderadmin/messaging/
        oaMsgRecipientGrps.xml"/>
    </REQUEST>
</REQUESTS>
```

The '`uploadMessagingSpec`' request takes a list of filenames and loads them. The path of the file can be absolute or relative to classpath.

## Message Events

Client services publish message events to the Messaging Service and notify it when an event occurs.

Example of ORDER_ADMIN service publishing events:

```
<messaging>
    <events>
        <event Name="CoCreditHoldEvent"
        ServiceName="ORDER_ADMIN" Document="CUSTOMER_ORDER"
                                  Category="Order"/>
        <event Name="AoRouteEvent" ServiceName="ORDER_ADMIN"
        Document="ACTIVITY_ORDER"
                                  Category="ActivityOrder"/>
        ...
    </events>
</messaging>
```

The '`ServiceName`' attribute above specifies the service that is publishing the event. The 'Document' attribute specifies the input document that the event is expecting. The event may or may not be defined with a document. If it is defined for a document then it is expected that the client service will pass the document when it raises (fires) a message event onto the messaging service. The document can be used to build the message content and to find the recipients. It is available as '`thisDoc`' variable in the messaging service. The '`Category`' attribute specifies the category that event belongs to. In future releases this will be used to group messages in different categories.

## Event Definition

The Event Definition specifies the actions to be performed by the messaging service when an event is received. An event can have one or more event actions. The event action has a name,

type and condition. The only type supported today is 'SEND_MESSAGE'. The condition is an xpath expression and if it is specified the event action will be executed only if the condition evaluates to true.

Event action of type 'SEND_MESSAGE' needs the message template to be used for the message content and the list of recipient groups to be used to figure out the recipients.

Example of Event Definition:

```
<messaging>
   <event_defs>
       <event_def Name="CoCreditHoldEvent"
       ServiceName="ORDER_ADMIN">
           <action Name="CreditHoldMessage" Cond="$thisDoc/
           ORDER_TYPE_ID/@Value = 'STANDARD'"
           Type="SEND_MESSAGE">
               <message_template Value="CoCreditHoldMT"/>
               <recipient_groups>
                   <recipient_group Value="CoCreatorRG"/>
                   <recipient_group Value="CoSalesPersonRG"/>
               </recipient_groups>
           </action>
       </event_def>
       <event_def Name="CoPromisedEvent"
       ServiceName="ORDER_ADMIN">
           <action Name="SendMessage1" Type="SEND_MESSAGE">
               <message_template Value="CoPromisedMT"/>
               <recipient_groups>
                   <recipient_group Value="CoOrderPointRG"/>
               </recipient_groups>
           </action>
       </event_def>
   </event_defs>
</messaging>
```

## Message Template

A message template defines the message contents for different communication protocols. The protocols that are currently supported are ALERT, EMAIL, HTTP, FILE and EAI. The message template need not define content for all five protocols.

Example of message template:

```
<messaging>
   <message_templates>
      <message_template_resource ServiceName="ORDER_ADMIN"
      ResourceBundleName="oaMsgResourceBundle"/>
      <message_template Name="CoCreditHoldMT"
      Document="CUSTOMER_ORDER" ServiceName="ORDER_ADMIN">
         <comm_protocol Type="ALERT">
             ... ALERT content. See details below ...
         </comm_protocol>
         <comm_protocol Type="EMAIL">
             ... EMAIL content. See details below ...
         </comm_protocol>
         <comm_protocol Type="HTTP">
             ... HTTP content. See details below ...
         </comm_protocol>
         <comm_protocol Type="FILE">
             ... FILE content. See details below ...
         </comm_protocol>
         <comm_protocol Type="EAI">
             ... EAI content. See details below ...
         </comm_protocol>
      </message_template>
   </message_templates>
</messaging>
```

### ALERT

Example of comm_protocol tag for type ALERT:

```
<comm_protocol Type="ALERT">
   <priority Value="5"/>
    <url>
       <document_id Select="$thisDoc/ID/@Value"/>
       <document_name Value="CUSTOMER_ORDER"/>
    </url>
```

```
    <message Type="TEXT">
        <text>Credit hold placed on customer order {0}
    with BillTo {1}</text>
        <args>
            <arg Select="$thisDoc/ID/@Value"/>
            <arg Select="$thisDoc/BILL_TO_ID/@Value"/>
        </args>
    </message>
</comm_protocol>
```

The 'priority' tag is optional and is only for info purpose only in this release. For an alert the 'document_id' and 'document_name' needs to be specified so that the UI can show a proper hyperlink for the alert. 'document_id' is an expression while 'document_name' has to be a fixed value. For the alert the UI will show the 'document_id' as a hyperlink to a page that can display document identified by 'document_name'. The actual text for the alert is also specified in the above specification. The 'message' tag is used for that purpose. The 'message' tag has a 'Type' attribute that specifies the type of message. The only 'Type' supported for ALERT protocol is 'TEXT'. The 'text' tag inside 'message' can be any text and can have arguments. Arguments will be identified by an index and will have an argument definition. The argument definition is an expression, which will be evaluated and substituted in the text at run-time. **message_template_resource** tag is used to specify the resource bundle name, which is used to localize the email messages.

**Note:** You have to specify the filename without extension. To localize all the e-mails to U.S English you need to have oaMsgResourceBundle_en_US.properties in your classpath.

To localize the XML messages you need to have localized xsl files (see "EMAIL" Section). The localized xsl files are loaded the same way java loads resource bundles.

For example if you specify i18n.xsl to localize for German you need to have i18n_de.xsl in your classpath.

### EMAIL

Example of comm_protocol tag for type EMAIL:

```
<comm_protocol Type="EMAIL">
    <subject Value="Credit hold on customer order"/>
    <from_address Select="$thisDoc/EMAIL/@Value"/>
    <format Value="text"/> <!-- valid values text|html -->
    <message Type="TEXT">
        <text>Credit check resulted in credit hold for Customer
        Order Id: {0}</text>
```

```
        <args>
            <arg Select="$thisDoc/ID/@Value"/>
        </args>
        </message>
</comm_protocol>


or


<comm_protocol Type="EMAIL">
    <subject Value="Credit hold on customer order"/>
    <from_address Select="$thisDoc/EMAIL/@Value"/>
    <format Value="text"/> <!-- valid values text|html -->
    <message Type="XML">
        <data Select="request('//ORDER_ADMIN/
        getEntireCOForEAI', 'ID', $thisDoc/ID/@Value)"/>
        <xsl_file Value="com/teradata/xservice/orderadmin/
        oa.xsl"/> <!-- Xsl file to translate xml returned by
        above 'data' expression to email body text -->
    </message>
</comm_protocol>
```

The '`subject`' tag above defines the subject to be used for emails.

The '`from_address`' tag specifies from address of Email. If there is no from_address is specified then default from address will be the system from address, which is in email.xml property file. `<param Name="om.systemFromAddress" Value="fromAdd@teradata.com"/>`

The '`format`' tag specifies the format of message. Valid values for this tag are "text" or "html". Default value is "html". If `<format Value="text"/>` is specified then message send in email will go as plain text i.e. there will not be any translation regarding html specific tags. This is useful when any special characters like new line, tab etc. are required in message but html is not supported by the email client. If `<format Value="html"/>` is specified and if there are any html specific tags, will get translated accordingly.

The '`message`' tag defines the body of the email. The '`message`' tag can be of two types: '`TEXT`' or '`XML`'. Both define a way to generate the body of the email. The '`data`' tag under message type '`XML`' is used to specify the xpath expression that will generate the required xml and the '`xsl_file`' tag defines the xsl file to be used to translate the xml generated by '`data`' to text or html that can be sent out as body of the email.

**Note:** For dynamic email subject, subject has to be given in this format in messageTemplates.xml

```
<subject Value="{0}">
<args>
<arg Select="$thisDoc/subject/@Value"/>
</args>
</subject>
```

### HTTP

Example of comm_protocol tag for type HTTP:

```
<comm_protocol Type="HTTP">
     <message Type="XML">
          <data Select="request('//ORDER_ADMIN/
          getShipNoticeData', $thisDoc)/SHIPMENTS"/>
     </message>
</comm_protocol>
```

For HTTP protocol only '`XML`' message type is allowed. This is similar to the '`XML`' message type described above. The '`Select`' attribute is a regular xpath expression as supported by xcore. The `XML` returned by the xpath expression is sent out as the message body to Integration service.

### FILE

Example of comm_protocol tag for type FILE:

```
<comm_protocol Type="FILE">
     <message Type="XML">
         <data Select="$thisDoc"/>
     </message>
</comm_protocol>
```

For FILE protocol the message type allowed is 'XML', similar to HTTP protocol.

### EAI

Example of comm_protocol tag for type EAI:

```
<comm_protocol Type="EAI">
     <message Type="XML">
         <data Select="$thisDoc"/>
     </message>
</comm_protocol>
```

This is similar to HTTP and FILE protocols.

## Recipient Groups

Recipient Group is a collection of recipients who will receive the message. A recipient group may be defined for a service and document or may be independent. Definition of recipient groups can have list of recipient definitions. Each recipient definition is evaluated at run-time and can result in collection of recipients.

Example of recipient group definition:

```
<messaging>
    <recipient_groups>
        <recipient_group Name="CoRG" Document="CUSTOMER_ORDER"
        ServiceName="ORDER_ADMIN">
            <recipient_def Name="RecipientDef1" Type="ENTITY">
                ... recipient def of type ENTITY. See details
            below ...
            </recipient_def>
            <recipient_def Name="RecipientDef2" Type="ROLE">
                ... recipient def of type ROLE. See details
            below ...
            </recipient_def>
        </recipient_group>
    </recipient_groups>
</messaging>
```

In the above example the recipient group 'CoRG' is defined for document 'CUSTOMER_ORDER' and service 'ORDER_ADMIN'. That means that the recipient group needs a CUSTOMER_ORDER document as defined by the ORDER_ADMIN service for evaluation. If the recipient group is defined for a document then it can only be used by those events that are defined for the same document. It is possible to have recipient groups that are not associated with any document/service.

Each recipient group can have a list of recipient definitions. In the above example the 'CoRG' recipient group has two recipient definitions 'RecipientDef1' and 'RecipientDef2'. The recipient definition has a 'Type' attribute. Valid types are: ENTITY, ROLE, RULE and 'CUSTOM'. The content of the recipient_def tag will vary depending on the 'Type'.

At run-time the messaging service will evaluate the recipient group by evaluating each recipient_def.

### ENTITY

Example of recipient_def for type ENTITY:

```
<recipient_def Name="Creator" Type="ENTITY">
     <entity_type Value="USER"/>
```

```
    <entity_id Select="$thisDoc/CREATED_BY_ID/@Value"/>
</recipient_def>
or
<recipient_def Name="OrderPoint" Type="ENTITY">
    <entity_type Value="OP"/>
    <entity_id Select="$thisDoc/ORDER_POINT_ID/@Value"/>
</recipient_def>
```

A recipient definition of type 'ENTITY' is used when an entity needs to receive the message. Entities referred here are the entities that are modeled by the EM service. Valid entities in EM are: USER, ORG (i.e. Organization), OP (i.e. Order Point), BT (i.e. BillTo), ST (i.e. ShipTo) and FC (i.e. Fulfillment Center). The recipient_def of type 'ENTITY' needs to have two properties defined: entity_type and entity_id. Entity_type should be one of the above listed values. The entity_id will be an expression, which will be evaluated at runtime and should evaluate to id of the entity modeled in EM service.

The EM service models all the entities and maintains their contact information like email address for users and url and export file path for other entities. The EM service also keeps the messaging preferences for each entity. Using the messaging preference the entities can control in what format they receive the messages. For more details check the EM service documentation. The EM API used by messaging service is getEntityMessagingInfo.

### ROLE

Example of recipient_def for type ROLE:

```
<recipient_def Name="Planner" Type="ROLE">
    <role_id Value="PL"/>
    <entity_type Value="OP"/>
    <entity_id Select="$thisDoc/ORDER_POINT_ID/@Value"/>
</recipient_def>
```

The recipient definition of type 'ROLE' evaluates to a list of recipients that have the specified role for given entity (domain). The roles and users association with roles/entities are maintained by the EM service. The EM API used by messaging service for role based recipient definitions is getRoleMessagingInfo. The roles supported by this API are:

| role_id | entity_type |
|---|---|
| 'CSR' (customer service representative) | 'ORG' (sell organization) |
| 'CR' (credit representative) | 'ORG' (sell organization) |
| 'PL' (planner) | 'ORG' (sell organization) |
| 'FM' (fulfillment manager) | 'FC' (fulfillment center) |

| role_id | entity_type |
|---|---|
| 'RA' (repair analyst) | 'ORG' (sell organization) |
| 'BYR' (buyer) | 'ORG' (sell/customer organization) |
| 'SA' (seller admin) | 'ORG' (sell organization) |
| 'CA' (customer admin) | 'ORG' (customer organization) |
| 'BMGR' (buyer manager) | 'OP' (order point) |
| 'LM' (location manager) | 'DC' (distribution center) |
| 'SUPPLIER' (supplier) | 'ORG' (sell organization) |
| 'DCMGR' (DC manager) | 'DC' (distribution center) |
| 'IMGR' (inventory manager) | 'LOC' (location) |

The '`entity_id`' property is an expression that at run-time should evaluate to id of entity specified by the '`entity_type`'.

### RULE

Example of recipient_def for type RULE:

```
<recipient_def Name="recipient1" Type="RULE">
     <request Name="customRuleForApproverMessagingInfo"
     ServiceName="APPROVAL"/>
</recipient_def>
```

This type of recipient definition is used when the logic to figure out the recipients is complex. Here the messaging service will call the user-defined rule to get the recipient information. The 'request' tag gives the name of the user-defined rule and the 'ServiceName' attribute gives the service that implements the rule. The input to the rule is the document (if a document is specified for the recipient group). The request to this rule will be:

```
<REQUEST Name="customRuleForApproverMessagingInfo "
ServiceName="APPROVAL">
     <!-- Document will be sent if it is available -->
     <CUSTOMER_ORDER>
          <ID Value="CO-1"/>
          <ORDER_POINT_ID Value="LOP-1"/>…
     </CUSTOMER_ORDER>
</REQUEST>
```

The output format expected back from the rule is as follows:

```
<RESPONSE>
```

```
<RECIPIENT>
<!-- Entity related info is optional -->
    <ENTITY_ID Value="ORG-1"/>
    <ENTITY_TYPE Value="ORG"/>
    <!--If EVENT_CATEGORIES tag is not present it will
    imply that the recipient is not interested in any
    events. A special event category 'All' is allowed -->
     <EVENT_CATEGORIES>
          <EVENT_CATEGORY Value="All"/>
     </EVENT_CATEGORIES>
     <!-- Only those protocols that the recipient is
   interested in need to be returned -->
     <COMM_PROTOCOLS>
         <COMM_PROTOCOL Type="EMAIL">
             <ADDR Value="abc@abc.com"/>
         </COMM_PROTOCOL>
         <COMM_PROTOCOL Type="ALERT">
              <USER_ID Value="bbabin"/>
         </COMM_PROTOCOL>
         <COMM_PROTOCOL Type="HTTP">
              <URL Value="http://localhost:3000/ordersIn"/
              >
         </COMM_PROTOCOL>
         <COMM_PROTOCOL Type="FF">
              <FILE Value="file:///c:/temp"/>
         </COMM_PROTOCOL>
         <COMM_PROTOCOL Type="EAI"/>
     </COMM_PROTOCOLS>
</RECIPIENT>
<RECIPIENT>
    ...
</RECIPIENT>
</RESPONSE>
```

### CUSTOM

Example of recipient_def for type CUSTOM:

```
<recipient_def Name="recipient1" Type="CUSTOM">
     <event_categories>
          <event_category Value="All"/>
     </event_categories>
     <comm_protocols>
          <comm_protocol Type="HTTP">
               <url Value="http://localhost:3000/ordersIn"/>
          </comm_protocol>
     </comm_protocols>
</recipient_def>
```

Many times there is a need to send messages to some fixed URL or email address. The custom recipient definition can be used in such cases. As sees above the custom recipient definition gives the event categories that recipient is interested in and the contact details of necessary protocols. The sample above specifies the URL for the HTTP protocol. This recipient will only get messages in HTTP format and the messages will be sent to the specified URL.

## RAISE MESSAGE EVENT

To raise a message event the client service uses following utility rule extension tag:

```
<RAISE_MSG_EVENT Name="createCoEvent" Select="$coDoc"
Synchronous="yes"/>
```

Here the '`Name`' is the name of the event that has already been registered with the messaging service. If the event is defined with a document then the document can be sent to the messaging service using the '`Select`' attribute. The Select is a regular xcore xpath select that should evaluate to xml document.

The '`Synchronous`' attribute is optional and controls as to whether the message is sent synchronously or not. The default behavior is to send the message in asynchronous manner.

# E-mail Service

E-mail service is one of the MDM services which provide capability to send email from MDM. E-mail service provides basic framework to send e-mail based on user events or as a result of workflow processing. E-mail service can be used for various scenarios such as notifying users for events, with status of application processing and sending necessary information.

## Usage

Below is the request invocation format of sendMail X-Rule and parameter information.

While invoking sendMail API, user can provide necessary information as below:

Table 103: SendMail API

| | |
|---|---|
| SMTP_HOST | SMTP host name for e-mail setup |
| FROM_ADDRESS | From Address for outgoing mails |
| TO_ADDRS | Address to send mail |
| CC_ADDRS | CC addresses |
| BCC_ADDRS | BCC addresses |
| SUBJECT | Subject line of e-mail |
| MESSAGE | Payload or detail message |
| ATTACHMENTS | Attachments to mail if any |

```
<REQUEST Name="sendEmail">
      <SMTP_HOST Value="teradata.com"/>
      <FROM_ADDRESS Value="dummy@teradata.com"/>
      <TO_ADDRS>
        <TO_ADDR Value="abc@teradata.com"/>
        <TO_ADDR Value="xyz@teradata.com"/>
      </TO_ADDRS>
      <CC_ADDRS>
        <CC_ADDR Value="fdhd@teradata.com"/>
        <CC_ADDR Value="ewe@teradata.com"/>
      </CC_ADDRS>
      <BCC_ADDRS>
        <BCC_ADDR Value="fdhd@teradata.com"/>
        <BCC_ADDR Value="ewe@teradata.com"/>
      </BCC_ADDRS>
      <SUBJECT Value="some subject"/>
      <FORMAT Value="TEXT or HTML"/>
      <MESSAGE>
        <DATA Value="some data"/>
```

```
        <XML_DATA XsltFileName="inv.xsl">
          ...
        </XML_DATA>
      </MESSAGE>
      <ATTACHMENTS>
        <ATTACH_FILE_NAME Value="demo.doc"/>
        <ATTACH_FILE_NAME Value="presentation.doc"/>
      </ATTACHMENTS>
</REQUEST>
```

# APPENDIX T Test Service

### What's In This Appendix

This appendix provides information on the Test service in MDM.

Topics include:


- Introduction
- Goals
- Test service setup


# Introduction

Test service, as the name suggests provides the framework to execute test requests on services deployed on a specified Teradata MDM server. The request and the associated response from the server can then be used to verify the result of the test. The results of the test are then written out to a file.

The service exploits the capability of Teradata MDM's workflow engine. Tests run as part of the test service, are essentially workflows. A test workflow can be viewed as a series of requests to the server, each request dependent on the previous one. Workflow acts as the glue that ties these requests through variables and provides for the flow of actions. Hence, every test workflow corresponds to testing a business workflow on the server through a series of API calls.

For example, Customer order creation and completion test would involve

1. Creation of customer order.
2. Acknowledgement of the resultant activity orders
3. Shipment notification
4. Invoice notification
5. Delivery notification

# Goals

The goals set for this service are:

1. Reduce the size of test workflows required to test a business functionality.
2. Provide a framework to verify the results of a test.

3 Provide a framework for reporting the results of tests.

4 Enable a high degree of reusability of test workflow code.

5 Provide a mechanism to instantiate test request data with as little manual intervention as possible.

6 Provide a common test infrastructure to execute unit tests, regression tests and PSR tests.

The sections below will provide the implementation and usage details of the attained goals.

# Test service setup

The framework files are present under

<MDM_Install_Directory>\ cfg\xservice\test.

## Configuration file

**`Configuration file:`** test.xml

This file is present at `<Install_Directory>\cfg\properties`. The configuration file contains the service config elements of xcore service setup.

Following are excerpts of the config file, which will need to be modified during setup.

*The testSetup element contains the information that the TEST service uses to initiate itself. The comments define what each property means and how it is used.*

```
<testSetup>
	<!-- Specify 'true' if you need to validate the test response. 'false' is usually used when the Test service is run for PSR testing. -->
	<DoExpectCheck Value="true" ValidValues="true|false"/>

	<!-- Default sleep time is the time the Test service waits after sending a request to the server  and before analysing the response. This feature is used to allow for asynch processes to finish on the server. .-->
	<SleepTime Value="10" Comment="In seconds"/>

	<!— Log Test Result specifies what needs to get reported as part of the Test result. "OnFail" is the default setting, which indicates that the results are to be reported in detail only upon a failure. "All" is the setting to report results in case of success or failure. "No" is set to not report the results at all and is usually used when the service is run for PSR testing.-->

	<LogTestResult Value="OnFail" ValidValues="OnFail|All|No"/>

	<!-- When LogTestData is set to true, the result file for each test workflow also contains the test request and the  response from the server.-->
	<LogTestData Value="false" ValidValues="true|false"/>

	<!-- TestSuiteResultDir specifies the directory in which the test results need to be logged. This directory needs to exist otherwise the service will throw an exception.-->
	<TestSuiteResultDir Value="../tests/results"/>
```

```
     <!-- This parameter is used to select the database sequence for
Id generation-->
        <DefaultTestIdDoc Value="TEST_DEFAULT"/>

</testSetup>
```

## Batch files

The <MDM_Install_Directory>/bin folder contains the files to run the service. You will need to modify the startServices.bat/sh file to invoke the test service from the xserver. Add the test service file, test.xml to this file. Execute the batch script runTests.bat/sh to post the test requests.

## Other files

TestRules.xml is present under <MDM_Install_Directory>\xservice\test\xml. New tests can be added under the data, workflow and expects directories at <MDM_Install_Directory>\cfg\xservice\toolkit\test. A detailed description of the files used for testing follows.

The log file for this service will be created in the directory specified in the config file. The log file has the usual log level settings available on any xcore service. If the “info” log level is turned on, the framework spits out a lot of detail during execution of the tests. These details include the test request, response, variable values, etc.

## Test files

Different types of files are used to execute tests. The example used here is customer order creation to completion test workflow.

### 1 Data templates

Data templates are specific to a test and provide the template upon which to construct the request data. Properties that need to be instantiated for every test run can be specified as xPath functions in the template. When the template is loaded, the functions are executed and the data value is set.

For example, following is a data template used to create a customer order. Note the functions (in bold) that have been used in the template to generate the test run dependent properties.

```
<CUSTOMER_ORDER>
   <EXT_ORDER_NUM Value="{uniqueDocumentId()()}"/>
   <ORDER_TYPE_ID Value="STANDARD" Editable="no"/>
   <ORDER_ORIGIN_ID Value="CSR"/>
   <CREATED_BY_ID Value="usr-91036642969436"/>
   <BEXT_ORDER_NUM Viewable="no" Editable="no"/>
   <SETTLEMENT_TYPE/>
```

```
   <ORDER_POINT_ID Value="SDUSER_00004L1R"/>

   <SALES_PERSON_ID Editable="yes" Viewable="yes"/>

   <CURRENCY_CODE Value="USD"/>

   <PRICING_TEMPLATE Value="183783DEB501441FB4E120F344D0CB19" Edit-
able="yes"/>

   <BILL_TO_ID Value="CDT_00004MME"/>

   <PAYMENT_TERMS/>

   <SHIP_TO_ID Value="CDT_00004MMD"/>

   <DEST_COUNTRY Value="USA"/>

   <TAXABLE Value="true" Editable="yes"/>

   <TAX_AUTHORITY Editable="yes"/>

   <TAX_EXEMPT_ID Editable="yes"/>

   <SHIP_MODE Value="GROUND"/>

   <SOURCE_FROM_ID Value="CB_DC_S"/>

   <ORDER_LINE_ITEMS>

      <ORDER_LINE_ITEM>

         <EXT_REF_ID Value="{uniqueDocumentId()}"/>

         <ORDER_TYPE_ID Value="STANDARD" Editable="yes"/>

         <LINE_ITEM_TYPE Value="REGULAR"/>

         <ITEM_ID Value="72610236"/>

         <ITEM_DESCR Value="MENS LOGO CREW IVY/BLACK L"/>

         <LIST_PRICE Value="9.0"/>

         <UNIT_PRICE Value="9"/>

         <CURRENCY_CODE Value="USD"/>

         <UOM Value="EA"/>

         <SELLING_ORG_ID Value="ORG_00004IOT"/>

         <SOURCE_FROM_ID Value="CB_DC_S"/>

         <BILL_TO_ID Value="CDT_00004MME"/>

         <DEST_COUNTRY Value="USA"/>

         <TAXABLE Value="true" Editable="yes"/>

         <TAX_AUTHORITY Editable="yes"/>

         <TAX_EXEMPT_ID Editable="yes"/>

         <MAX_NUM_DELIVERIES Value="3"/>

         <SHIP_TERMS/>

         <DEFAULT_PRICER_TERMS Value="200"/>

         <NATURE_OF_STOCK_MOVEMENT Value=""/>
```

```
<ORDER_REQUESTS>
   <ORDER_REQUEST>
      <EXT_REF_ID Value="{uniqueDocumentId()}"/>
      <LINE_ITEM_ID Value="li-43_1043768947252"/>
      <UOM Value="EA"/>
      <UNIT_PRICE Value="9"/>
      <CURRENCY_CODE Value="USD"/>
      <BILL_TO_ID Value="CDT_00004MME"/>
      <DEST_COUNTRY Value="USA"/>
      <SHIP_TERMS/>
      <REQ_QTY Value="10"/>
      <DELIVERY_DATE Value="{incrDate(date(), duration(1,0,0,0))}"/>
      <MAX_NUM_DELIVERIES Value="3"/>
      <PROMISING_POLICY Value="ASAP"/>
      <SHIP_VIA Value="U"/>
      <SHIP_TO_ID Value="CDT_00004MMD"/>
   </ORDER_REQUEST>
</ORDER_REQUESTS>
<OVRD_PRICER_TERMS Value="200"/>
<APPLY_TERMS_OVRD Value="False"/>
<OVRD_LIST_PRICE Value="9"/>
<APPLY_PRICE_OVRD Value="False"/>
<LINE_NOTE Value=""/>
<LINE_TAG Value=""/>
<ROUTING_CODE Value=""/>
<SHIP_VIA Value="U"/>
<CRESTING_CHARGES/>
<ORDERED_QTY Value="10"/>
<QTY_PER Value="10"/>
<SHIP_TO_ID Value="CDT_00004MMD"/>
<PAYMENT_INST_ID Value="bc-12221036519763468"/>
<PAYMENT_INST_TYPE Value="BC"/>
<SUPPLIER_ID Value="TaylorMade"/>
</ORDER_LINE_ITEM>
</ORDER_LINE_ITEMS>
```

### 2 Expects

Expect files are written for each specific test in a test workflow. In the expect files, the test writer can specify what the expected result of a particular test should be. The files usually correspond to an API request/response. However, the same expect file can be reused in other tests, if the expected response is the same. Also, there can be any number of expects specified for a given test.

The test request is used as the input to the expect file. The framework runs through the test request and as each tag is encountered it executes the defined asserts for each encountered tag. The following example outlines how the expect file is to be defined. Explanations for each tag are specified as comments.

The features to note in the following specification are

**a** Pre test variables

**b** Post test variables

**c** What are tags and what are fields

**d** Variable definition at a tag level

**e** Field definition by denoting it as any represented by ".".

**f** Assert result reporting by specifying the comment with dynamic property values.

**g** Error levels that can be specified: Fail, Abort.

**h** Termination of expect verification when an assert with error level "Abort" fails.

For example:

```
<expect InputDesc="The request data for createCustomerOrder">

  <!-- pre_test_variables are computed before the execution of a TEST.
The scope of these variables is limited to this expect.-->

  <pre_test_variables>

     <!-- $testRequest variable is available -->
     <variable Name="inputCoDoc" Exprn="$testRequest/REQUEST/CUSTOM-
ER_ORDER"/>

     <variable Name="inputLineCount" Exprn="count($inputCoDoc/OR-
DER_LINE_ITEMS/ORDER_LINE_ITEM)"/>

  </pre_test_variables>

      <!-- post_test_variables are computed after the execution of a
      TEST. The scope of these variables is limited to this expect. -->

  <post_test_variables>

       <!-- $testResponse, $testRequest variable is available -->
     <variable Name="responseCoDoc" Exprn="$testResponse/RESPONSE/
CUSTOMER_ORDER"/>

  </post_test_variables>

  <!-- This spec is executed after the test is run and the post test
variables are evaluated. The input to this spec is the test request.
The TEST engine runs through the test request element by element and
```

```
when the encountered tag has an assert defined, the assert is executed.
The variables defined as part of the assert spec are computed as and
when they are used. -->

   <assert_spec>
      <tags>
         <tag Name="CUSTOMER_ORDER">

   <variables>
            <variable Name="dbCoDoc" Exprn="request('//ORDER_ADMIN/
getEntireCustomerOrder', 'CUSTOMER_ORDER_ID', $testResponse/RE-
SPONSE/CUSTOMER_ORDER/ID/@Value)/CUSTOMER_ORDER"/>
            <variable Name="createdLineCount" Exprn="count($dbCoDoc/
ORDER_LINE_ITEMS/ORDER_LINE_ITEM)"/>
            <variable Name="numberOfLinesCreated" Exprn="count($db-
CoDoc/ORDER_LINE_ITEMS/ORDER_LINE_ITEM)"/>
         </variables>

     <!-- Excecute the following asserts when the encountered tag is
CUSTOMER_ORDER-->
         <asserts>
            <!-- Every assert has a name, an xpath expression that
evaluates to true/false and an error level.
            The error levels are: Abort, Fail. Abort results in ter-
mination of further expect processing if the assert evaluates to
false.
            The assert also has on_true and on_false report elements
that are used for reporting the results.
            -->
            <assert Name="Validate createCustomerOrder response" Ex-
prn="$testResponse/RESPONSE/CUSTOMER_ORDER/ID/@Value != null" Error-
Level="Abort">
               <on_true Report="Customer order created with Id
{$testResponse/RESPONSE/CUSTOMER_ORDER/ID/@Value}"/>
               <on_false Report="Customer order was not created"/>
            </assert>
            <assert Name="Validate number of lines" Exprn="$created-
LineCount = $inputLineCount">
               <on_true Report="All lines created"/>
               <on_false Report="All lines were not created"/>
            </assert>
         </asserts>
     <!-- Tag and Field elements are akin to document and properties.
When the tag element CUSTOMER_ORDER is encountered, the asserts spec-
ified for CUSTOMER_ORDER are executed, followed by the asserts spec-
ified for its fields/properties. Every child element of CUSTOMER_ORDER
qualifies as a field element. -->
         <fields>
         <!-- The "." field element evaluates to any child element
of CUSTOMER_ORDER-->
            <field Name=".">
               <!-- A filter is specified in the assert to evaluate
only those field elements that have a Value attribute.-->
               <assert Name="Valid property assert" Filter="$this-
Field/@Value != null" Exprn="findProp($dbCoDoc, name($thisField))/
_ERRORS/@Value = null">
                  <on_true Report="valid prop"/>
         <!-- To specify dynamic properties in the report, use the
syntax {$variableName}-->
                  <on_false Report="CUSTOMER_ORDER property
{name($thisField)} with value {$thisField/@Value} has an error with
error code {findProp($dbCoDoc, name($thisField))/_ERRORS/_ERROR/@Val-
ue}"/>
               </assert>
```

```
            </field>
          </fields>
        </tag>
        <tag Name="ORDER_LINE_ITEM">
          <variables>
            <variable Name="extRefId" Exprn="$thisDoc/EXT_REF_ID/
@Value"/>
            <variable Name="dbLiDoc" Exprn="$dbCoDoc/OR-
DER_LINE_ITEMS/ORDER_LINE_ITEM[EXT_REF_ID/@Value = $extRefId]"/>
            <variable Name="createdRequestCount" Exprn="count($this-
Doc/ORDER_REQUESTS/ORDER_REQUEST)"/>
            <variable Name="dbRequestCount" Exprn="count($dbLiDoc/
ORDER_REQUESTS/ORDER_REQUEST)"/>
          </variables>
          <asserts>
            <assert Name="Line has been created" Exprn="$dbLiDoc !=
null">
              <on_true Report="Order line item with Ext Ref Id {$ex-
tRefId} has been created with Id {$dbLiDoc/ID/@Value}"/>
              <on_false Report="Order line item with Ext Ref Id {$ex-
tRefId} has not been created"/>
            </assert>
            <assert Name="Validate number of requests" Exprn="$cre-
atedRequestCount = $dbRequestCount">
              <on_true Report="All order requests created"/>
              <on_false Report="All order requests were not creat-
ed"/>
            </assert>
          </asserts>
          <fields>
            <field Name=".">
              <assert Name="Valid property assert" Filter="$this-
Field/@Value != null" Exprn="findProp($dbLiDoc, name($thisField))/
_ERRORS/@Value = null">
                <on_true Report="valid prop"/>
                <on_false Report="ORDER_LINE_ITEM property
{name($thisField)} of value {$thisField/@Value} has an error with er-
ror code {findProp($dbCoDoc, name($thisField))/_ERRORS/_ERROR/@Val-
ue}"/>
              </assert>
            </field>
          </fields>
        </tag>
      </tags>
    </assert_spec>
</expect>
```

### 3 Workflows

A test workflow is used to execute a series of test requests. The data required for the requests are loaded as part of the workflow actions, the variables are initialized and set and the workflow can call upon sub-workflows to perform reusable component actions.

The current test framework does not support workflows that have event nodes. So, please avoid using event nodes in your test workflows.

Following is an example of a test workflow. The two actions tags to note in the example are:

PROCESS_TEMPLATE

TEST

Other features to note are:

**a** Test tag requires a 'REQUESTS' child tag and 'EXPECTS' child tag.

**b** The expect file can be run in two modes, by specifying the Value attribute. If the value attribute says 'true', the asserts are expected to evaluate to 'true'. If the value attribute says 'false', the asserts are expected to evaluate to 'false'. This change in direction increases the reusability of an expect specification.

**c** Note that the test workflows can invoke APIs on several services

For example:

```
<workflows>

   <workflow Name="OrderCreateAndComplete">

      <variables>
         <variable Name="coId" Type="string"/>
         <variable Name="successIf" Type="string"/>
      </variables>
      <parameter_types>
         <parameter_type Name="XMLform"/>
      </parameter_types>

      <nodes>

         <start Name="ConstructData">
            <actions>
               <action Name="SpecifyCreateOrderRequestDetails">


               <!-- PROCESS_TEMPLATE is a xrule action tag. It loads the
specified template data file, executes the XPath functions specified
in the data file and assigns the XMLform object instance to the vari-
able specifed. If there were functions mentioned in the template file,
these will be executed and the values set.-->

                  <PROCESS_TEMPLATE FromValue="StandardOrder-
With2Lines.xml" AssignToVar="coDoc"/>


               </action>
            </actions>

            <next_nodes OutputVar="coDoc" OutputType="XMLform">
               <next_node Name="CreateOrder"/>
            </next_nodes>
         </start>

         <decision Name="CreateOrder" InputVar="coDoc" InputType="XML-
form">
            <actions>
               <action Name="CallCreate">

               <!-- The TEST action tag corresponds to a single API test.
                  Name attribute specifies the name of the test.
                  TEST tag requires a child REQUESTS tag. The REQUESTS
tag contains the request to the server. Every child tag of REQUESTS
```

```
will be treated as plain XML element.
                TEST tag also requires a child EXPECTS tag. The expects
tag contains the EXPECT elements where the expect file names are spec-
ified.
            -->

               <TEST Name="CreateCustomerOrderTest">
                  <REQUESTS ServiceName="ORDER_ADMIN" Timing-
Stats="2">
                     <REQUEST Name="tmag_createCustomerOrder" Servi-
ceName="ORDER_ADMIN">
                        <TO_XML DocVar="coDoc"/>
                     </REQUEST>
                  </REQUESTS>
                  <EXPECTS>
                     <EXPECT File="CreateCustomerOrderExpect.xml"
Value="true"/>
                     <EXPECT File="PromisingExpect.xml" Value="true"/
>
                  </EXPECTS>
               </TEST>
            </action>
         </actions>
         <condition Value="$testResponse/RESPONSE/TEST_RESULT/@Val-
ue = 'Pass'"/>
         <if_true>
            <next_nodes OutputVar="testResponse" OutputType="XML-
form">
               <next_node Name="SetTheVariables"/>
            </next_nodes>
         </if_true>
         <if_false>
            <next_nodes>
               <next_node Name="Done"/>
            </next_nodes>
         </if_false>
      </decision>
      <task Name="SetTheVariables" InputVar="response" Input-
Type="XMLform">
         <actions>
            <action Name="variables">
               <SET Var="coId" FromSelect="getFirstDoc('//ORDER_AD-
MIN/CUSTOMER_ORDER', 'EXT_ORDER_NUM', $response/RESPONSE/CUSTOM-
ER_ORDER/EXT_ORDER_NUM/@Value)/ID/@Value"/>
               <SET Var="successIf" FromValue="true"/>
            </action>
         </actions>
         <next_nodes>
            <next_node Name="AcknowledgeAllAos"/>
         </next_nodes>
      </task>

      <!-- This call to a sub-workflow shows how reusability can be
introduced in Test workflows. The AcknowledgeAllActivityOrders sub-
workflow can be called from several different workflows to execute the
function of acknowledging activity orders. In this call, you can also
specify the expect files that are to be used allowing for the sub-
workflow to be branded as a functional template ignorant of data or
result.-->

      <sub_workflow Name="AcknowledgeAllAos" Workflow="Acknowled-
geAllActivityOrders">
         <pre_actions>
            <action Name="data">
```

```
                  <TO_DOCVAR AssignToVar="data">
                     <DATA>
                        <CO_ID Value="{$coId}"/>
                        <ACK_TYPE Value="ACCEPT"/>
                        <ACTIVITY_ORDERS Value="ALL"/>
                        <EXPECTS>
                           <EXPECT File="AcknowledgeActivityOrderEx-
pect.xml" Value="{$successIf}"/>
                        </EXPECTS>
                        <FOR_CHANGE Value="false"/>
                     </DATA>
                  </TO_DOCVAR>
               </action>
            </pre_actions>
            <input>
               <TO_XML DocVar="data"/>
            </input>
            <next_nodes>
               <next_node Name="ShipTheOrder"/>
            </next_nodes>
         </sub_workflow>
         <sub_workflow Name="ShipTheOrder" Workflow="SentShip-
mentTest">
            <pre_actions>
               <action Name="data">
                  <TO_DOCVAR AssignToVar="data">
                     <DATA>
                        <CO_ID Value="{$coId}"/>
                        <ORDER_SCHEDULES Value="ALL"/>
                        <EXPECTS TestName="SentShipmentTest">
                           <EXPECT File="SentShipmentExpect.xml" Val-
ue="true"/>
                        </EXPECTS>
                     </DATA>
                  </TO_DOCVAR>
               </action>
            </pre_actions>
            <input>
               <TO_XML DocVar="data"/>
            </input>
            <next_nodes>
               <next_node Name="LoadExternalInvoice"/>
            </next_nodes>
         </sub_workflow>
         <sub_workflow Name="LoadExternalInvoice" Workflow="LoadExter-
nalInvoiceTest">
            <pre_actions>
               <action Name="data">
                  <TO_DOCVAR AssignToVar="data">
                     <DATA>
                        <CO_ID Value="{$coId}"/>
                        <ORDER_SCHEDULES Value="ALL"/>
                        <EXPECTS TestName="LoadInvoiceTest">
                           <EXPECT File="LoadInvoiceExpect.xml" Val-
ue="true"/>
                        </EXPECTS>
                     </DATA>
                  </TO_DOCVAR>
               </action>
            </pre_actions>
            <input>
               <TO_XML DocVar="data"/>
            </input>
            <next_nodes>
```

```
            <next_node Name="CheckFinalOrderState"/>
          </next_nodes>
        </sub_workflow>
        <task Name="CheckFinalOrderState">
          <actions>
            <action Name="OrderState">
              <TEST Name="VerifyOrderProperties">
                <REQUESTS ServiceName="ORDER_ADMIN">
                  <GET_DOCUMENT Name="CUSTOMER_ORDER">
                    <ID Value="{$coId}"/>
                  </GET_DOCUMENT>
                </REQUESTS>
                <EXPECTS>
                  <EXPECT File="FinalOrderExpect.xml" Val-
ue="true"/>
                </EXPECTS>
              </TEST>
            </action>
          </actions>
          <next_nodes>
            <next_node Name="Done"/>
          </next_nodes>
        </task>
        <done Name="Done"/>
      </nodes>
    </workflow>
    </workflows>
```

### 4 Requests

One request file is used to specify the workflows that need to be executed. RUN_TEST_WORKFLOWS command is used to execute the test workflows specified.

For example:

```
<REQUESTS ServiceName="TEST">

   <RUN_TEST_WORKFLOWS>

      <TEST_WF Name="OrderCreateAndCompletedWith6LinesAndFooter" Ser-
viceName="TEST"/>
      <TEST_WF Name="OrderCreateAndComplete" ServiceName="TEST"/>
      <TEST_WF Name="OrderCreateAndCompletedWith6LinesAndFooter" Ser-
viceName="TEST"/>
      <TEST_WF Name="OrderCreateAndChange" ServiceName="TEST"/>
      <TEST_WF Name="OrderCreateAndRepromise" ServiceName="TEST"/>

   </RUN_TEST_WORKFLOWS>

   </REQUESTS>
```

### 5 Results

Result files are generated upon each run of the Test request. Every test workflow will have a corresponding result file named, {WorkflowName}.xml. In addition to this, there will be a file called OverallResults.xml that comprises of the list of workflows run with the corresponding result namely, ‘Pass’ or ‘Fail’.

For example: Following is content of file OrderCreateAndComplete.xml

```
<TEST_SUITE Name="OrderCreateAndComplete" Result="Fail">
```

```
<TEST_CASES>

<!-- Following were the TESTs that were part of the workflow.-->
   <TEST_CASE Name="CreateCustomerOrderTest" Result="Pass">

      <!-- The results of the test: CreateCustomerOrder-->
      <RESULTS>

         <ASSERTS ExpectFile="D:\OMxViews

         <ASSERTS ExpectFile="D:\OMxViews

      </RESULTS>
   </TEST_CASE>

   <TEST_CASE Name="SentShipmentTest" Result="Fail">
      <RESULTS>
         <ASSERTS ExpectFile="D:\OMxViews

            <ASSERT Name="Check for shipment in DB" Result="Fail">
               <REPORT Value="Shipment has not been created"/>
            </ASSERT>
         </ASSERTS>
      </RESULTS>
   </TEST_CASE>

   <TEST_CASE Name="SentShipmentTest" Result="Fail">
      <RESULTS>
         <ASSERTS ExpectFile="D:\OMxViews

            <ASSERT Name="Check for shipment in DB" Result="Fail">
               <REPORT Value="Shipment has not been created"/>
            </ASSERT>
         </ASSERTS>
      </RESULTS>
   </TEST_CASE>

   <TEST_CASE Name="SentShipmentTest" Result="Fail">
      <RESULTS>
         <ASSERTS ExpectFile="D:\OMxViews

            <ASSERT Name="Check for shipment in DB" Result="Fail">
               <REPORT Value="Shipment has not been created"/>
            </ASSERT>
         </ASSERTS>
      </RESULTS>
   </TEST_CASE>

   <TEST_CASE Name="LoadInvoiceTest" Result="Pass">
      <RESULTS>
         <ASSERTS ExpectFile="D:\OMxViews

      </RESULTS>
   </TEST_CASE>

   <TEST_CASE Name="VerifyOrderProperties" Result="Pass">
      <RESULTS>
         <ASSERTS ExpectFile="D:\OMxViews

      </RESULTS>
   </TEST_CASE>
</TEST_CASES>
</TEST_SUITE>
```

For example: Of OverallResults.xml

```
<OVERALL_RESULTS Result="Fail">

  <TEST_SUITE Name="OrderCreateAndCompletedWith6LinesAndFooter" Re-
sult="Fail" DetailsFile="OrderCreateAndCompletedWith6LinesAndFoot-
er.xml"/>

      <TEST_SUITE Name="OrderCreateAndComplete" Result="Fail" De-
tailsFile="OrderCreateAndCompleteWf.xml"/>

  </OVERALL_RESULTS>
```

## Dictionary of Xpath functions and Xcore actions specific to Test service

1 **containsString()**

Usage:

```
<COND Value=" containsStr($dbReq/STATE_HISTORY/@Value,
'PROMISED')"/>
```

This function returns true if the specified string (that is. PROMISED) is present in the string value that results from the evaluation of `($dbReq/STATE_HISTORY/@Value)`.

2 **fileToXml()**

Usage:

```
<SET Var="fileData" FromSelect="fileToXml('fileName')"/>
```

This function loads the XML in the specified file, instantiates an XMLform object and associates the variable name with this value. This function will not evaluate any functions that have been specified in a data file.

3 **findProp()**

Usage:

```
<SET Var="error" FromSelect="findProp($dbCoDoc,
name($thisField))/_ERRORS/@Value"/>
```

This function locates the property of the XMLForm referred to by `$dbCoDoc` with the same name as `$thisField`.

4 **getDocInstances()**

Usage:

```
<SET Var="docs" FromSelect="getDocInstances('ORDER_ADMIN',
$notesForDoc, 'EXT_REF_ID', $notesForDocExtRefId)"/>
```

This function retrieves document instances. It differs from `getDocument()` in that it allows a variable document name.

5 **getMap()**

Usage:

```
<SET Var="schedAoDocMap" FromSelect="getMap($dbResponse/
ORDER_SCHEDULE,'AO_ID')"/>
```

This function groups document instances based on a specified property value. In this example, ORDER_SCHEDULE documents are grouped based on the value of property AO_ID.

The grouped result is assigned to schedAoDocMap which will have the following structure:

```
<MAP>
    <KEY Name="AO_ID" Value="AO-02920">
        <ORDER_SCHEDULE>
            <!-- ORDER SCHEDULE DATA-->
        </ORDER_SCHEDULE>
        <ORDER_SCHEDULE>
            <!-- ORDER SCHEDULE DATA-->
        </ORDER_SCHEDULE>
    </KEY>
    <KEY Name="AO_ID" Value="AO-6832658">
        <ORDER_SCHEDULE>
            <!-- ORDER SCHEDULE DATA-->
        </ORDER_SCHEDULE>
        <ORDER_SCHEDULE>
            <!-- ORDER SCHEDULE DATA-->
        </ORDER_SCHEDULE>
    </KEY>
</MAP>
```

6 **uniqueDocumentId()**

Usage:

```
<EXT_ORDER_NUM Value="{uniqueDocumentId()}"/>
```

This function sets the value of this property with a uniquely generated Id.

7 **getDateTime()**

Usage:

```
<SET Var="dateTime" FromValue="{getDateTime(incrDate(date(),
duration(3,0,0,0)))}"/>
<SET Var="dateTime" FromValue="{getDateTime()}"/>
```

This function converts timestamp into date-time format. It takes as arguments nothing in which case it returns current date time else it takes a date object in which case it converts it into date time format and returns the string.

8 **PROCESS_TEMPLATE**

Usage:

```
<PROCESS_TEMPLATE FromValue="StandardOrderWith2Lines.xml"
AssignToVar="coDoc"/>
```

This action tag loads the XML data from the file, instantiates an XMLform object and runs through the object to execute functions to generate the Value attribute. It then assigns the value to the variable specified.

**9 APPEND_TO_XML**

This action tag works like a TO_DOCVAR action tag. It appends the XML to a specified node.

Usage:

```
     <APPEND_TO_XML SelectList="$shipmentDoc/ACTIVITY_ORDERS/AC-
            TIVITY_ORDER" IsLiteral="false">
          <ID Value="{$shipGroup/@Value}"/>
          <ORDER_LINE_ITEMS>
               <ORDER_LINE_ITEM>
                  <ORDER_REQUESTS>
                     <ORDER_REQUEST>
                        <ORDER_SCHEDULES>
                           <FOR_EACH SelectList="$shipGroup/OR-
DER_SCHEDULE" CurrentElement="sched">
                              <TO_DOCVAR AssignToVar="newSched">
                                 <ORDER_SCHEDULE>
                                    <ID Value="{$sched/ID/@Value}"/>
                                    <SHIPPED_QTY Value="{$sched/QTY/
@Value}"/>
                                    <SHIP_DATE Value="{$shipmentDoc/
SHIP_DATE/@Value}"/>
                                 </ORDER_SCHEDULE>
                              </TO_DOCVAR>
                              <TO_XML Select="$newSched"/>
                           </FOR_EACH>
                        </ORDER_SCHEDULES>
                     </ORDER_REQUEST>
                  </ORDER_REQUESTS>
               </ORDER_LINE_ITEM>
          </ORDER_LINE_ITEMS>
  </APPEND_TO_XML>
```

**10 SET_NODE_VALUE**

Usage:

```
<SET_NODE_VALUE Node="$shipmentDoc/EXT_ID"
Value="{$shipmentDoc/ID/@Value}"/>
```

This action tag is like a SET action tag but it allows for the variable to be computed through an expression.

**11 POLL**

This action tag executes a specified condition till the condition evaluates to true or till the maximum count specified is reached.

If the condition does not evaluate to true at the end of maximum number of executions, the value of LOG_ON_FAILURE is logged in the service's log file. The result of the poll is assigned to the result variable. The value can be 'Pass' or 'Fail'

```
  <POLL MaxCount="35" ResultVar="pollResult" Cond="getFirstDoc('//
ORDER_ADMIN/ACTIVITY_ORDER', 'CO_ID', $coId, 'STATE_ID', 'PEND-
ING_ROUTING') = null">

     <LOG_ON_FAILURE Value="No activity orders in SFF state for CO
with Id {$coId}"/>
```

```
</POLL>
```

**12 TEST**

This action tag is used to run a test. The TEST action tag corresponds to a single API test. Name attribute specifies the name of the test.

TEST tag requires a child REQUESTS tag. The REQUESTS tag contains the request to the server. Every child tag of REQUESTS will be treated as plain XML element. TEST tag also requires a child EXPECTS tag. The 'expects' tag contains the EXPECT elements where the expect file names are specified.

The logic in the TEST action tag is the following.

**a** The test request data is generated.

**b** All the Pre-test variables specified in each of the expect files are evaluated and kept.

**c** The request is sent to the xcore server through the Poster class.

**d** The response from the xcore server is obtained.

**e** The executing thread is then made to sleep for the time specified in the configuration file. This allows for asynch processes on the server to complete.

**f** The test request and response are made available in the context.

**g** The Post-Test variables defined in the expect files are evaluated.

**h** The framework runs through the test request and as each tag is encountered, it evaluates the asserts defined for this tag. The asserts defined for this tag can be scattered across each of the expect files.

**i** As and when the asserts are evaluated, reports are generated and kept in the context.

**j** After all the asserts have been evaluated, the result reports are written to the result file. If any exception occurs during processing, the verification is terminated and the results are written to the result files.

**k** The test result value ("Pass" or "Fail" ) is set in the test response variable as a child. Tag TEST_RESULT.

**l** The workflow in which this TEST is defined now has access to 3 variables.

**i** Test request through `$testRequest`

**ii** Test response through `$testResponse`

**iii** Test result through `$testResponse/RESPONSE/TEST_RESULT/@Value`

```
<TEST Name="CreateCustomerOrderTest">
     <REQUESTS ServiceName="ORDER_ADMIN" TimingStats="2">
          <REQUEST Name="tmag_createCustomerOrder" ServiceN-
          ame="ORDER_ADMIN">
               <TO_XML DocVar="coDoc"/>
          </REQUEST>
     </REQUESTS>

     <EXPECTS>
          <EXPECT File="CreateCustomerOrderExpect.xml" Val-
```

```
                ue="true"/>
                <EXPECT File="PromisingExpect.xml" Value="true"/>
            </EXPECTS>
        </TEST>
```

**13 TEST_DATA**

Usage:

This action tag behaves similar to TEST action tag but it does not execute a request on the server. It merely uses the data specified to run an expect verification. This action tag becomes useful when the expect for a particular test has to be evaluated at a time long after the execution of the test request.

```
            <TEST_DATA Name="Verify committed data">

                <DATA>
                  <TO_XML Select="$changeRequest"/>
                </DATA>

                <TO_XML SelectList="$expects/EXPECTS"/>
</TEST_DATA>
```

For example: A change to a customer order request is sent to the server. You do not want to evaluate the results right then, but you want to acknowledge all the activity orders and then evaluate the system conditions for the change request. Following is the workflow specification for such a requirement.

```
<workflows>
   <workflow Name="ChangeAndAckSubWf">
      <variables>
         <!-- variable definition -->
      </variables>
      <nodes>
         <start Name="Set data">
            <actions>
               <action Name="Set data">
                  <!-- Some actions -->
               </action>
            </actions>
            <next_nodes>
               <next_node Name="ChangeTest"/>
            </next_nodes>
         </start>
         <task Name="ChangeTest">
            <actions>
               <action Name="Test">
                  <!-- The test request is executed at this point  and
the response is validated against a simple expect check that just con-
firms that the request was successfully executed.-->
                  <TEST Name="{$testName}">
                     <REQUESTS ServiceName="ORDER_ADMIN">
                        <TO_XML Select="$changeRequest"/>
                     </REQUESTS>
                     <EXPECTS>
                        <EXPECT File="SimpleChangeExpect.xml" Val-
ue="true"/>
                     </EXPECTS>
                  </TEST>
               </action>
            </actions>
            <next_nodes OutputVar="testResponse" OutputType="XMLform">
               <next_node Name="DecideStepAfterChange"/>
```

```
            </next_nodes>
         </task>
         <decision Name="DecideStepAfterChange" Input-
Var="testResponse" InputType="XMLform">
            <actions>
               <action Name="Decide">
                  <PRINT Value="***** testResponse from initial change"
Nl="yes"/>
                  <PRINT Var="testResponse" Nl="yes"/>
               </action>
            </actions>
            <condition Value="$testResponse/RESPONSE/TEST_RESULT/@Val-
ue = 'Pass'"/>
            <if_true>
               <next_nodes>
                  <next_node Name="AckAos"/>
               </next_nodes>
            </if_true>
            <if_false>
               <next_nodes>
                  <next_node Name="Done"/>
               </next_nodes>
            </if_false>
         </decision>

         <!-- Some other actions are performed -->
         <sub_workflow Name="AckAos" Workflow="AcknowledgeAllActivi-
tyOrders">
            <pre_actions>
               <action Name="data">
                  <!-- Some actions -->
               </action>
            </pre_actions>
            <input>
               <TO_XML DocVar="data"/>
            </input>
            <next_nodes>
               <next_node Name="DoChangeExpectCheck"/>
            </next_nodes>
         </sub_workflow>

         <!-- The request from the previous TEST is used to validate
the system data -->

         <task Name="DoChangeExpectCheck">
            <actions>
               <action Name="Check">
                  <TEST_DATA Name="Verify committed data">
                     <DATA>
                        <TO_XML Select="$changeRequest"/>
                     </DATA>
                     <TO_XML SelectList="$expects/EXPECTS"/>
                  </TEST_DATA>
               </action>
            </actions>
            <next_nodes>
               <next_node Name="Done"/>
            </next_nodes>
         </task>
         <done Name="Done"/>
      </nodes>
   </workflow>
   </workflows>
```

## Steps to run PSR tests

1. Specify the default thread count under psrSetup in the test.xml configuration file.
2. On the xcore command prompt of the TEST xcore server, type command loadTest. This command takes following as arguments.
   a. Arg1: numberOfThreads
   b. Arg2: numberOfRequestsPerThread
   c. Arg3: filename

## FAQ

1. *Do I have to write workflows for every test that I want to automate?*

   Yes, a workflow is the basic unit that is required for automating a test. If a test is a simple "ADD_DOCUMENT" request, then you will still need to use a workflow.

2. *Can I use 'TEST' tag in an xrule?*

   Currently, the test framework supports expect verification and result reporting only if the TEST is part of a workflow. This is because we are defining a workflow as the minimum pre-requisite to using this test framework.

3. *Can I pass parameters to start a test workflow?*

   Parameters are not allowed to be passed to an initiating test workflow. The initiating test workflow is meant to be an independent unit that can execute a test on its own. The sub workflows this initiating workflow calls upon can have input parameters.

4. *Can I specify how many ever expect files for a test?*

   Yes, there is no limitation to the number of expect files.

5. *Are the variables I define in one expect file (either as pre-test variables or post-test variables) available in another expect file for the same test?*

   No, pre and post-test variables have a scope limited to the expect file they are defined in.

6. *At what point are the variables defined in an expect file computed/evaluated?*

   Pre-Test and Post-Test variables are evaluated at all times. Variables defined at the tag level are evaluated at the point they are being used. Tag variables that are defined in an expect file but never used, will not get evaluated at all.

7. *How can I find out what the values are for the variables?*

   Turn on the log level "Info" for the TEST service. This will result in logging of the variable values as and when they get evaluated.

8. *My test request does not have a field/ tag that I need to verify for the test. The test response however has the field/tag. How do I define an assert for such a field/tag?*

   For a tag that is absent in the test request, you will always have a parent tag ('REQUESTS' is the root tag of all test requests). You can define an assert under this parent tag. This assert can ensure that the response has the tag that you expect.

   For a field that is absent in the test request, you can define an assert for it under its parent tag.

Example, STATE_ID of CUSTOMER_ORDER is absent in the test request and I expect the value of the field to be ‘COMPLETED’ in the response. You can define an assert that checks for STATE_ID value when the CUSTOMER_ORDER tag is encountered in the test request.

**9** *I have a specific assert defined for a child “A” of a parent “X”. But for all the other children of parent “X” I want to use the “.” Feature to define asserts. How does the test framework evaluate the asserts in this case?*

Asserts defined for a particular field specified with the field name has higher precedence compared to the assert defined for fields with a generic “.” name.

So, if you define an assert for field “A” and also define asserts with generic “.”, the asserts defined for “.” will not be executed for the field “A”. Only the assert defined for “A” will be executed.

**10** *As a corollary to the previous question, what if I do not want to perform an assert evaluation for a field but I want to use the “.” feature?*

If you define an empty assert like in the example below, field mentioned in the empty assert will be ignored and all other fields of the parent tag will be verified.

```
<fields>
   <!-- This is an empty assert definition for the field CHANGE_TYPE
-->
   <field Name="CHANGE_TYPE"/>

   <!-- CHANGE_TYPE will be ignored when evaluating this generic as-
sert-->
   <field Name=".">
      <assert Name="Valid property assert" Filter="($CHANGE_TYPE/@Val-
ue = 'CHANGE') and ($thisField/@Value != null)" Exprn="findProp($ex-
istingLine, name($thisField))/_ERRORS/@Value = null"/>
      <on_true Report="valid prop"/>
      <on_false Report="Line item property {name($thisField)} with
value {$thisField/@Value} has an error with error code {findProp($ex-
istingLine, name($thisField))/_ERRORS/_ERROR/@Value}"/>

   </field>
</fields>
```

# APPENDIX U MDM_SYS_ENT_STATE

### What's In This Appendix

This appendix provides details about the life cycle management of Master Data for data maintenance using SYS_ENT_STATE attribute provided by Teradata MDM in master staging entities.

Topics include:


- Introduction
- SYS_ENT_STATE Attribute
- MDM SYS_ENT_STATE Business Use Case


## Introduction

Teradata MDM helps you manage key data elements within the Enterprise Data Warehouse (EDW), within an Active Data Warehouse (ADW), as well as, across various enterprise systems and geographies. Teradata MDM enables this with the following high-level key features:

- Data staging for loading, cleansing, and validation.
- Master data life cycle management for data maintenance and business specific workflows and processes.

The MDM user can update this attribute as and when required based on the flow of entities in the MDM system to control the life cycle of an entity within the MDM system.

## SYS_ENT_STATE Attribute

Teradata MDM provides default system properties to each of the Master entities at the time of schema generation or incremental schema generation. You need not define the system default properties (SYS_SOURCE, SYS_CREATED_BY, SYS_CREATION_DATE, SYS_ENT_STATE ('ACTIVE','INACTIVE','NEW','DELETED'), SYS_LAST_MODIFIED_BY, SYS_LAST_MODIFIED_DATE etc.) on Master entities for data maintenance. Also, you can define any custom property on MDM Master entities, if required through the MDM Studio. For details, refer to *MDM Platform Studio User Guide.pdf.*

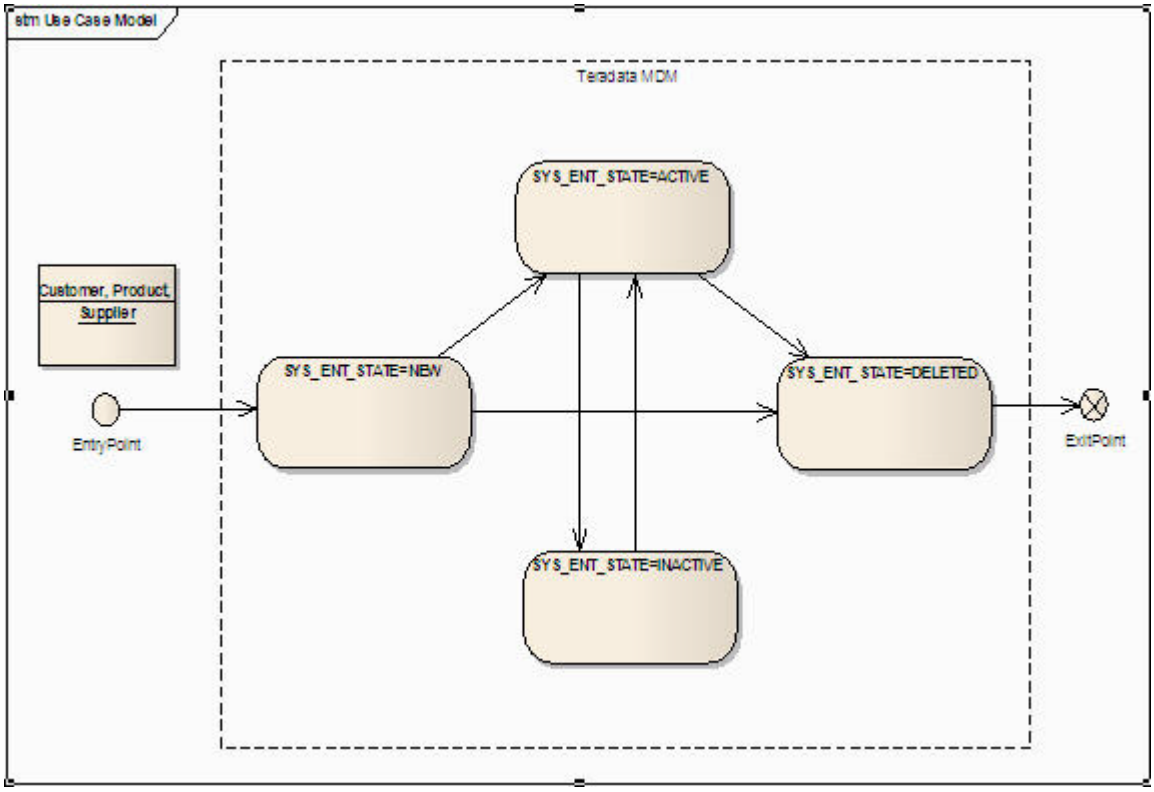
MDM uses SYS_ENT_STATE attribute to manage the life cycle of master data within the system. The SYS_ENT_STATE attribute notifies the state of the instance of the entity in the

MDM system. Default valid values for SYS_ENT_STATE are NEW, ACTIVE, INACTIVE and DELETED.

| SYS_ENT_STATE | Description |
|---|---|
| NEW | Entity is created but still not ready to be published/consumed. |
| ACTIVE | Entity is ready to be published or consumed. |
| INACTIVE | Entity is inactivated for business reasons—ready to be activated at later point in time. |
| DELETE | Entity is marked as deleted and non-existent in UI and the MDM system. |

The Figure 107 displays the state transition of an instance (Record) of an entity within the Teradata MDM system.

Figure 107: State Transition Model




The Figure 107 describes the following:

1 When the record is introduced to the MDM system, the state of the instance would be NEW.

2 If required, the business rules are set and if validated, then the state will be changed to ACTIVE and the record would be in the MDM system and available for consumption or publication.

3 If the underlying business rules fails, then the state of the record can be changed to DELETED and it will no longer be available in the MDM system.

4 An ACTIVE record can be made INACTIVE based on business rules for the specific period of time in the MDM system and the record will not be processed but will be available in the MDM system later point of time.

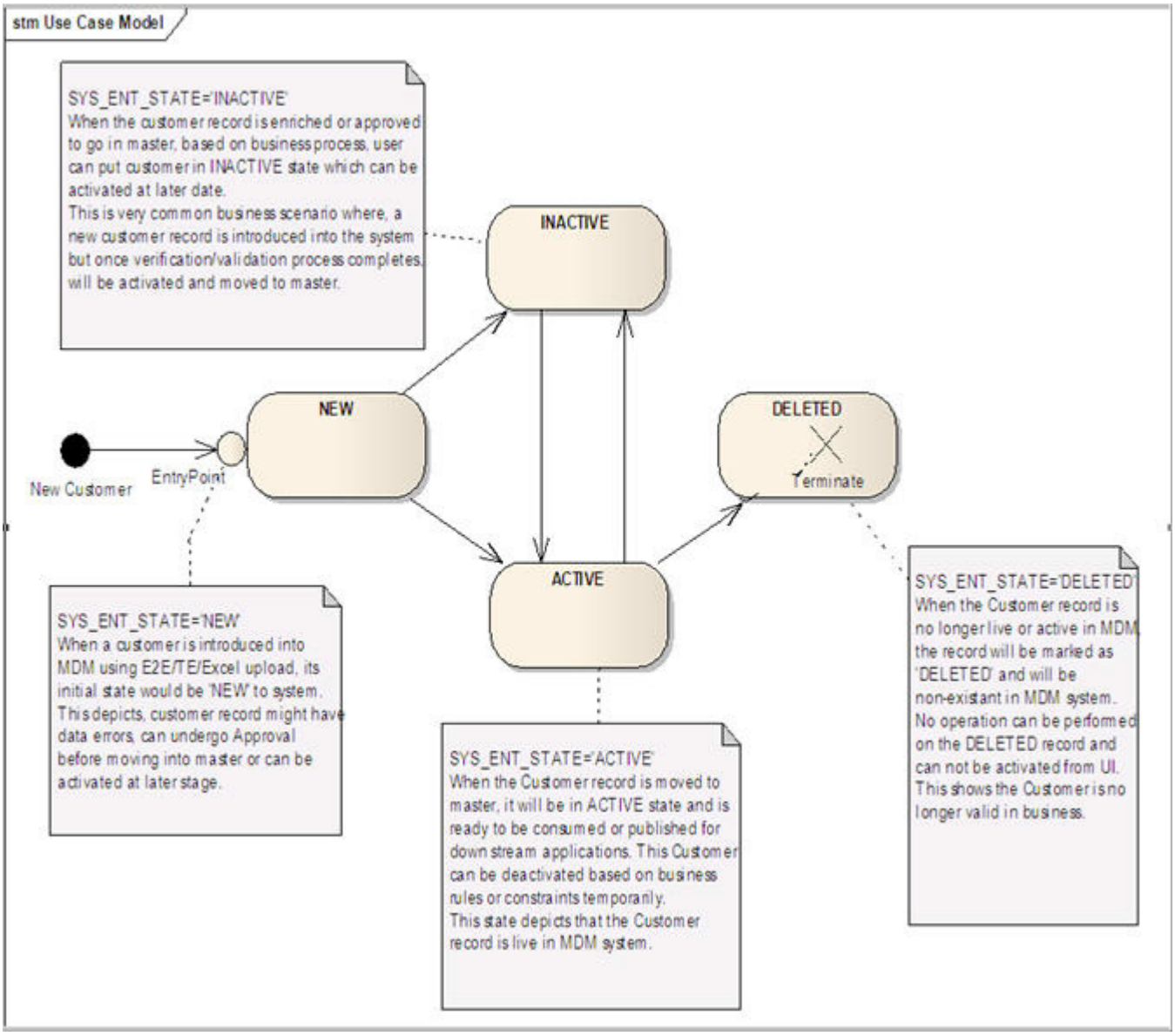
5 An INACTIVE record can be made ACTIVE based on business rules after the specific period of time and the record will be available to the MDM system for processing.

6 If an ACTIVE record is no longer valid in the MDM system then the record will be moved to DELETED state and it will not be available for the processing and considered to be non-existent in the MDM system.

# MDM SYS_ENT_STATE Business Use Case

**Use Case 1:** A new customer introduced into CDI MDM. Figure 108 describes the state transition model of a new customer introduced into the CDI MDM.

Figure 108: State Transition Model



# APPENDIX V Multiple Table Updates using DocView

### What's In This Appendix

This appendix provides guidelines for multiple table updates from UI.

Topics include:


- Introduction
- Multiple Tables from DocView


## Introduction

For multiple table updates, defining a view in Studio involving all the required columns of tables and then perform schema generation. Once the schema is generated, you can view those columns as a part of an integrated table. This approach is limited to viewing purpose only and to update the view, you need to navigate go to underlying individual tables and update them separately.

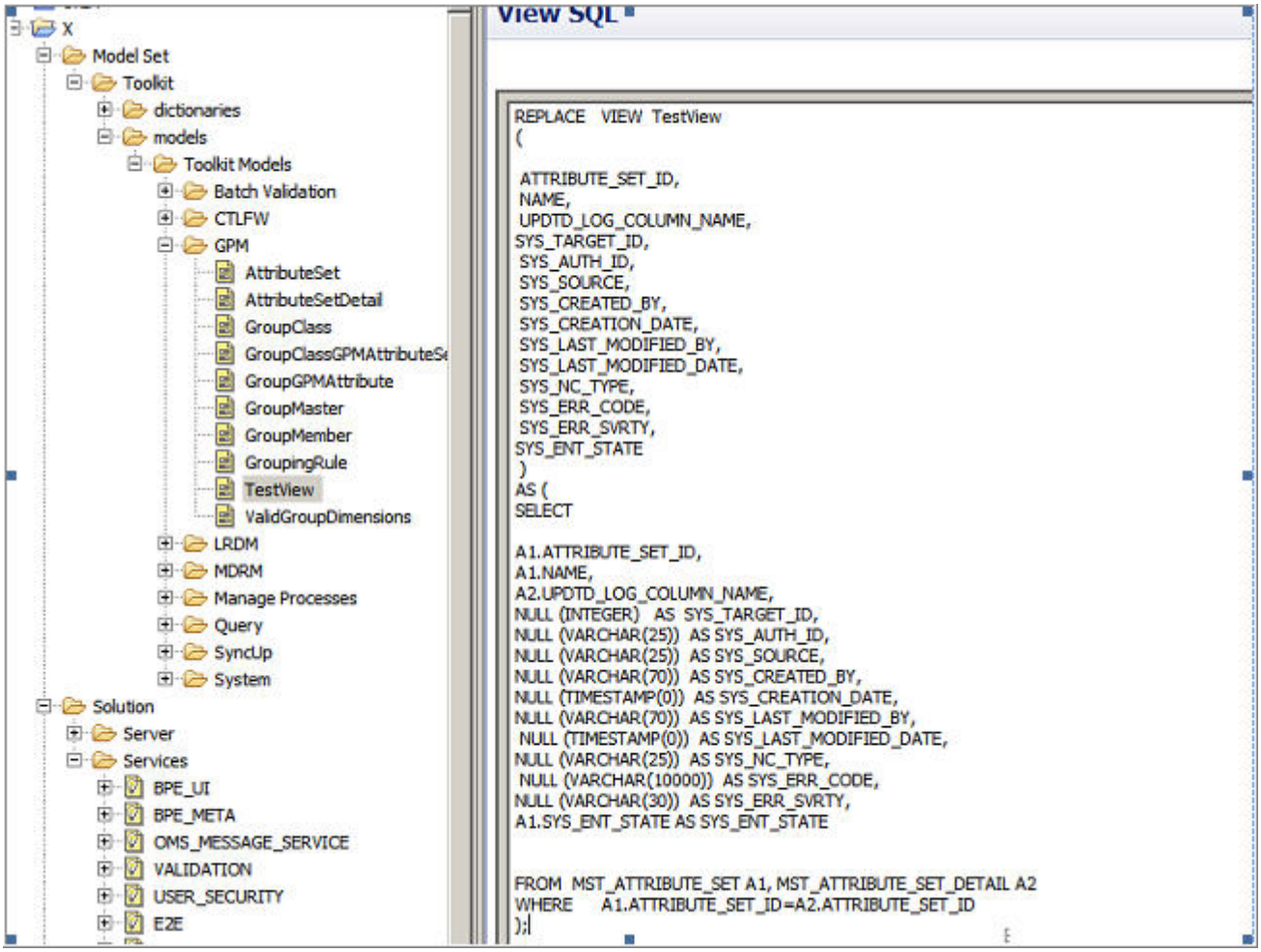
This document provides a design approach to update the data in the view using the DO_DB_PERSIST request using custom rule. You can create the required custom rules regarding updating the data in the view and these rules can be called after reading the required flag in the DO_DB_PERSIST call.

## Multiple Tables from DocView

Perform the following steps to update multiple tables from docview:

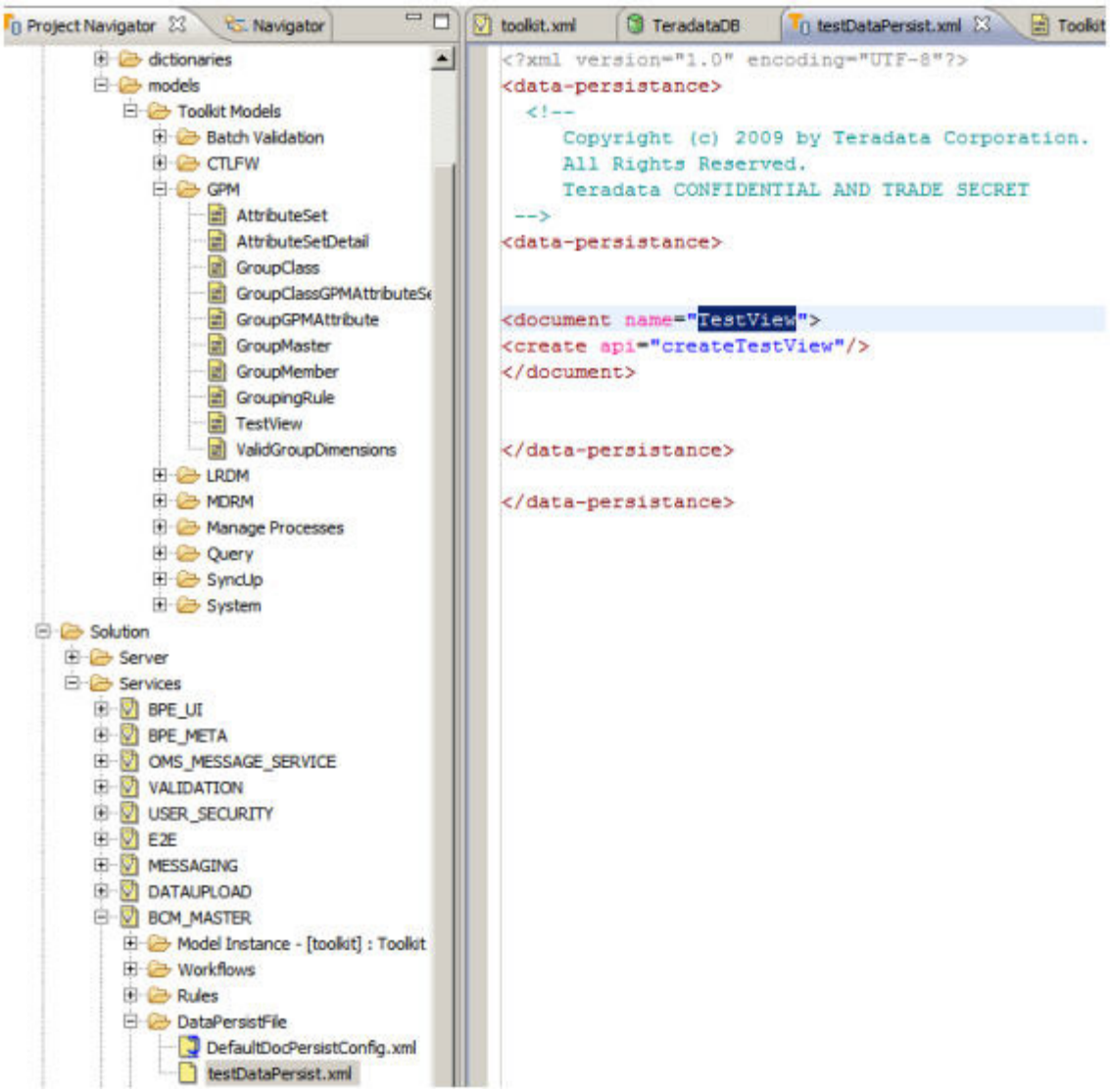
1 Create a view involving multiple tables, for example create a view named TestView using under the model GPM as in Figure 109.

Figure 109: Create View




2 Create a new data persist file in the service of choice as in Figure 110.

Figure 110: Create Data Persist File

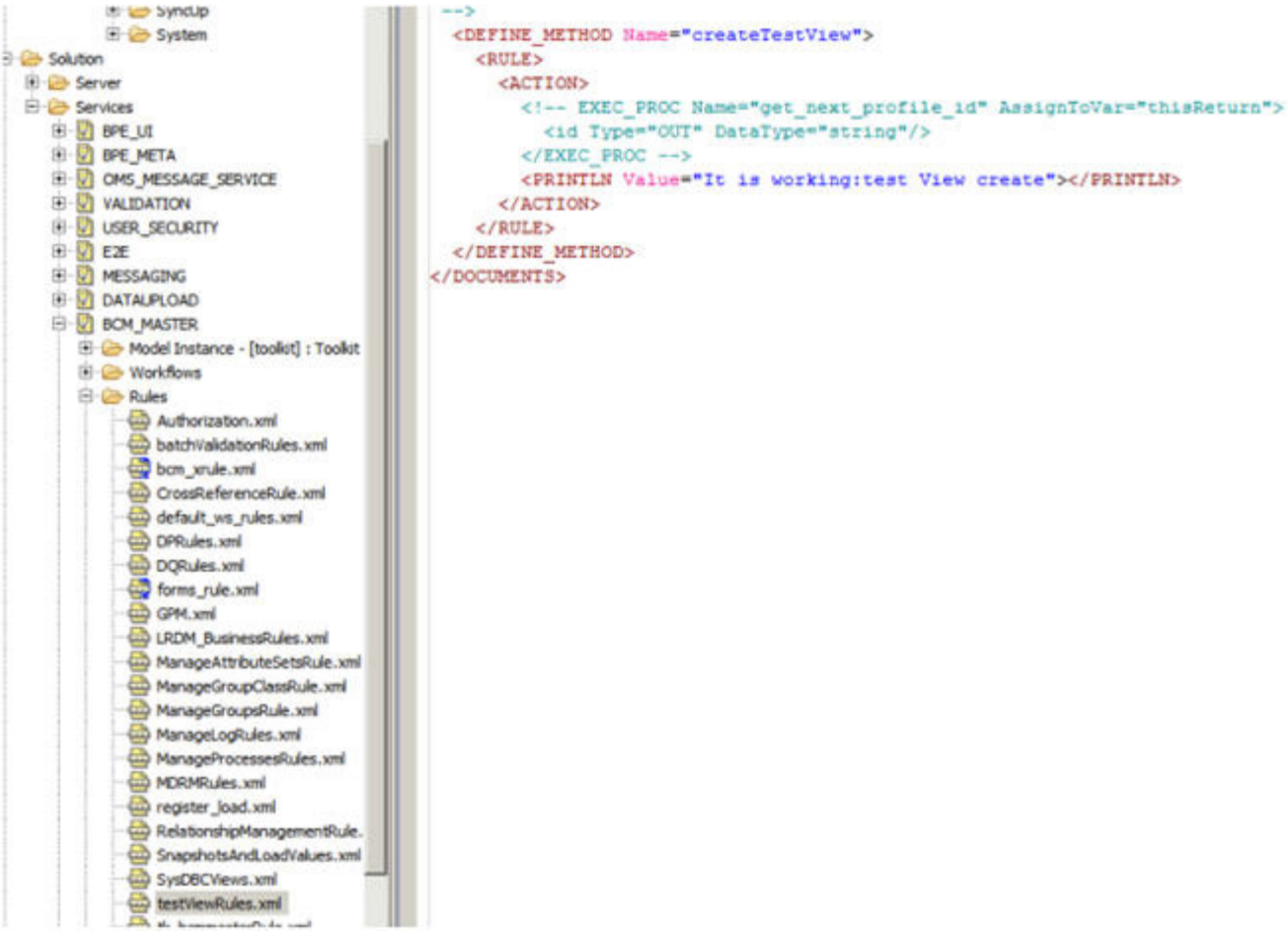



3 In the newly created data persist file, add the created view name (example: testview) in the document name tag.

On the Docview created, you can perform create, update, delete and mass update actions.

4 Create custom x-rules to perform the actions (create, update, delete and mass update) mentioned in the data persist file as in Figure 111.

Figure 111: Create Rule




**5** Restart the servers and you can start using the views from UI.

# APPENDIX W Parent-Child Validation

### What's In This Appendix

This appendix provides implementation steps for parent child validation.

Topics include:


- Introduction
- Parent-Child Validation


# Introduction

The following sections lists the steps to be performed by PGL developers to implement Parent-Child validation.

For example considers two dropdowns namely 'Country' and 'State'. The Values for 'State' dropdown should be populated based on selected value in parent 'Country'.

# Parent-Child Validation

The Parent-Child Validation as known as Multi value/key support. Perform the following implement parent-child validation:

1. Consider two columns Country and State existing in .mtt, which point to code sets Country and State respectively. Country is the parent code set for State.
2. Add two new attributes to PGL <ui:dropdown> namely 'codeSet' and 'parentCodeSet' as in the below sample code.

```
<TO_DOCVAR AssignToVar="temp">
  <ui:dropdown xmlns:ui="http://www.teradata.com/ui"
      sort="true" displayText="{$columnProperties/displayName/@Value}" id="{$tabName}.{$colName}"
      name="{$tabName}.{$colName}" title="{$columnProperties/hoverText/@Value}" required="{$required}"
      selectOne="true" optionSrc="$root/DATA/LOOKUP_DATA/{$tabName}.{$colName}/*" disabled="{$editable='false'}"
      codeSet="{$columnProperties/IsValidValue/@Value}"
      parentCodeSet="{$parentCodeSet/*/NAME/@Value}"/>
</TO_DOCVAR>
```

If value for 'codeSet' is 'State'. The value for 'parentCodeSet' should be 'Country'. It is expected that another dropdown exist with 'codeSet' attribute value 'Country'.

**Note:** Get value for 'parentCodeSet' attribute using below query.

```
LOCKING ROW FOR ACCESS SELECT PARENT_CODE_SET.NAME FROM MST_CODE_SET
CODE_SET,MST_CODE_SET PARENT_CODE_SET where
CODE_SET.PARENT_CODE_SET_ID=PARENT_CODE_SET.CODE_SET_ID AND
CODE_SET.NAME='{$columnProperties/IsValidValue/@Value}';
```

3 Add attribute 'parent' for each <ui:option> which can be populated using <GET_VALID_VALUES> is enhanced to return 'parent' for each value.

<GET_VALID_VALUES> is enhanced to return 'parent' for each value. A section of <GET_VALID_VALUES> result is given below.

```
<VALID_VALUE Value="YEM" Description="YEM" Parent="COUNTRY"/>
<VALID_VALUE Value="YUG" Description="YUG" Parent="COUNTRY"/>
<VALID_VALUE Value="ZAF" Description="ZAF" Parent="COUNTRY"/>
<VALID_VALUE Value="ZMB" Description="ZMB" Parent="COUNTRY"/>
<VALID_VALUE Value="ZWE" Description="ZWE" Parent="COUNTRY"/>
</PROPERTY>
<PROPERTY Name="State_Ext" Document="PARTY_MAILING_ADDRESS">
<VALID_VALUE Value="ARG.BA" Description="ARG.BA" Parent="ARG"/>
<VALID_VALUE Value="ARG.C" Description="ARG.C" Parent="ARG"/>
<VALID_VALUE Value="ARG.CBA" Description="ARG.CBA" Parent="ARG"/>
<VALID_VALUE Value="ARG.CF" Description="ARG.CF" Parent="ARG"/>
<VALID_VALUE Value="ARG.CHO" Description="ARG.CHO" Parent="ARG"/>
<VALID_VALUE Value="ARG.CHT" Description="ARG.CHT" Parent="ARG"/>
<VALID_VALUE Value="ARG.CTS" Description="ARG.CTS" Parent="ARG"/>
<VALID_VALUE Value="ARG.ER" Description="ARG.ER" Parent="ARG"/>
<VALID_VALUE Value="ARG.F" Description="ARG.F" Parent="ARG"/>
<VALID_VALUE Value="ARG.J" Description="ARG.J" Parent="ARG"/>
<VALID_VALUE Value="ARG.LP" Description="ARG.LP" Parent="ARG"/>
<VALID_VALUE Value="ARG.LR" Description="ARG.LR" Parent="ARG"/>
<VALID_VALUE Value="ARG.MS" Description="ARG.MS" Parent="ARG"/>
<VALID_VALUE Value="ARG.MZA" Description="ARG.MZA" Parent="ARG"/>
<VALID_VALUE Value="ARG.N" Description="ARG.N" Parent="ARG"/>
<VALID_VALUE Value="ARG.RN" Description="ARG.RN" Parent="ARG"/>
<VALID_VALUE Value="ARG.S" Description="ARG.S" Parent="ARG"/>
<VALID_VALUE Value="ARG.SC" Description="ARG.SC" Parent="ARG"/>
<VALID_VALUE Value="ARG.SE" Description="ARG.SE" Parent="ARG"/>
<VALID_VALUE Value="ARG.SF" Description="ARG.SF" Parent="ARG"/>
```

For example: Consider below dynamic dropdown

```
<ui:dropdown sort="true" displayText="State_Ext"
id="PARTY_MAILING_ADDRESS.State_Ext" name="PARTY_MAILING_ADDRESS.State_Ext"
title="State_Ext" selectOne="true"
optionSrc="$root/DATA/LOOKUP_DATA/PARTY_MAILING_ADDRESS.State_Ext/*"
disabled="false" defaultValue="" codeSet="STATE"
parentCodeSet="COUNTRY"/>
```

Options should be constructed from <GET_VALID_VALUES> result as below.

```
  <OPTION Value="VEN" Title="VEN" Parent="COUNTRY"/>
  <OPTION Value="VGB" Title="VGB" Parent="COUNTRY"/>
  <OPTION Value="VIR" Title="VIR" Parent="COUNTRY"/>
  <OPTION Value="VNM" Title="VNM" Parent="COUNTRY"/>
  <OPTION Value="VUT" Title="VUT" Parent="COUNTRY"/>
  <OPTION Value="WLF" Title="WLF" Parent="COUNTRY"/>
  <OPTION Value="WSM" Title="WSM" Parent="COUNTRY"/>
  <OPTION Value="YEM" Title="YEM" Parent="COUNTRY"/>
  <OPTION Value="YUG" Title="YUG" Parent="COUNTRY"/>
  <OPTION Value="ZAF" Title="ZAF" Parent="COUNTRY"/>
  <OPTION Value="ZMB" Title="ZMB" Parent="COUNTRY"/>
  <OPTION Value="ZWE" Title="ZWE" Parent="COUNTRY"/>
</PARTY_MAILING_ADDRESS.Country_Ext>
<PARTY_MAILING_ADDRESS.State_Ext tabNamePhy="MST_PARTY_MAILING_ADDRESS">
  <OPTION Value="ARG.BA" Title="ARG.BA" Parent="ARG"/>
  <OPTION Value="ARG.C" Title="ARG.C" Parent="ARG"/>
  <OPTION Value="ARG.CBA" Title="ARG.CBA" Parent="ARG"/>
  <OPTION Value="ARG.CF" Title="ARG.CF" Parent="ARG"/>
  <OPTION Value="ARG.CHO" Title="ARG.CHO" Parent="ARG"/>
  <OPTION Value="ARG.CHT" Title="ARG.CHT" Parent="ARG"/>
  <OPTION Value="ARG.CTS" Title="ARG.CTS" Parent="ARG"/>
  <OPTION Value="ARG.ER" Title="ARG.ER" Parent="ARG"/>
  <OPTION Value="ARG.F" Title="ARG.F" Parent="ARG"/>
  <OPTION Value="ARG.J" Title="ARG.J" Parent="ARG"/>
  <OPTION Value="ARG.LP" Title="ARG.LP" Parent="ARG"/>
```

4 HTML generation is shown below for each <ui:dropdown> and <ui:option>

This is generated by MDM framework and nothing is required to be done by PGL developers.

**Country dropdown**

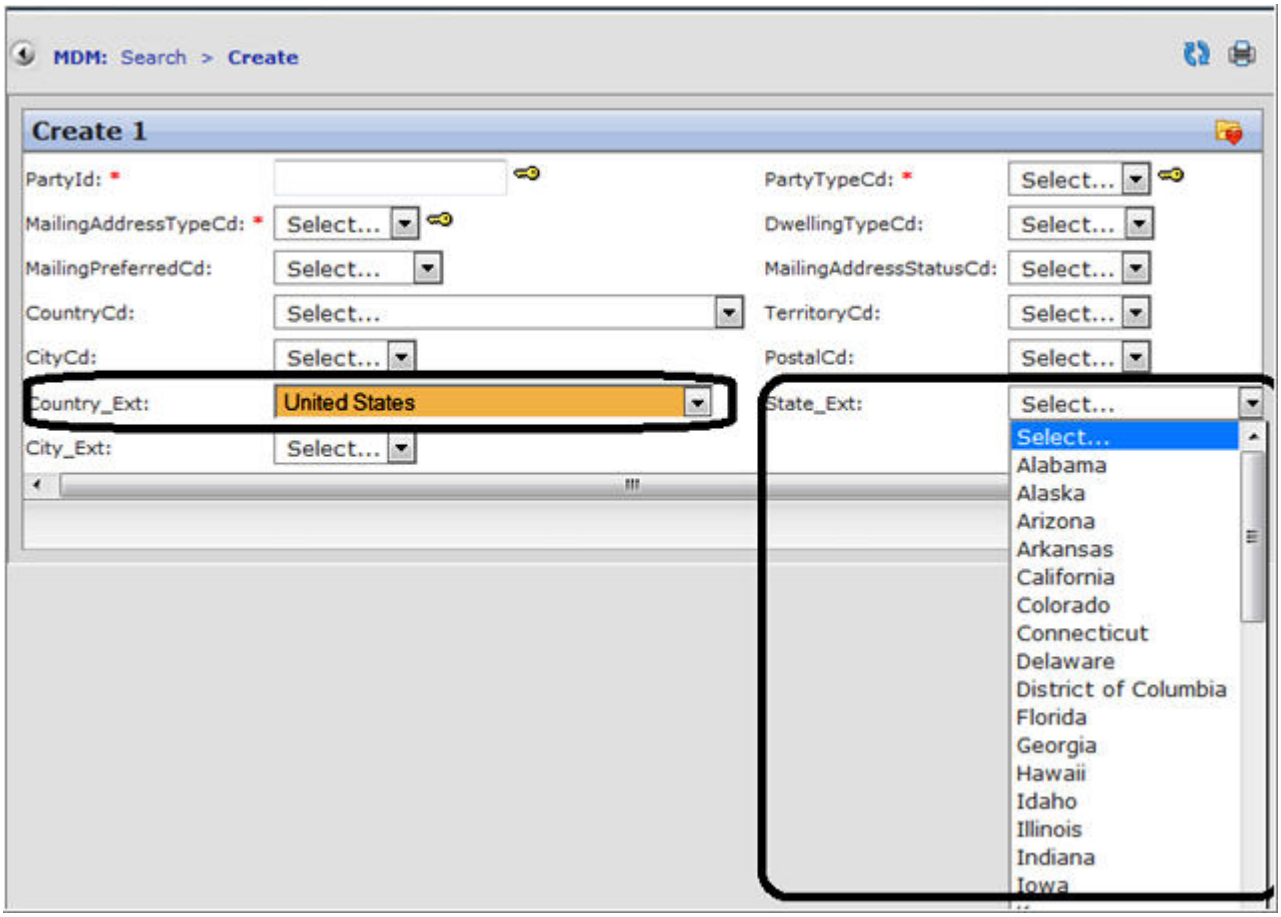
```
<select id="PARTY_MAILING_ADDRESS.CountryCd"
class="pulldown" data-parent-
code-set="Categories" data-code-set="COUNTRY"
title="CountryCd"
onfocus="javascript:setOriginalValue(this)"
onchange="javascript:isValid_field_if_data(this);
onChangeValue(this)"
name="PARTY_MAILING_ADDRESS.CountryCd"
containerid="step1">
<option value=""></option>
<option data-parent="COUNTRY"
title="Afghanistan" value="AFG"></option>
<option data-parent="COUNTRY" title="Albania"
value="ALB"></option>
<option data-parent="COUNTRY" title="Algeria"
value="DZA"></option>
<option data-parent="COUNTRY" title="American
Samoa" value="ASM"></option>
```

**State dropdown**

```
<select id="PARTY_MAILING_ADDRESS.State_Ext"
class="pulldown" data-parent-code-set="COUNTRY"
data-code-set="STATE" title="State_Ext"
onfocus="javascript:setOriginalValue(this)"
onchange="javascript:isValid_field_if_data(this);
onChangeValue(this)"
name="PARTY_MAILING_ADDRESS.State_Ext"
containerid="step1">
<option value=""></option>
<option data-parent="GBR" title="Aberdeen"
value="GBR.AB"></option>
<option data-parent="BRA" title="Acre"
value="BRA.AC"></option>
<option data-parent="ITA" title="Agrigento"
value="ITA.AG"></option>
<option data-parent="MEX"
title="Aguascalientes" value="MEX.AGS"
```

5 UI looks as shown below. Values for 'State_Ext' dropdown are enabled / disabled based on parent 'Country_Ext' dropdown selection. This logic is handled at framework level using new JavaScript function (pgl_v3.js file) populateChildren(obj).

This is generated by MDM framework and nothing is required to be done by PGL developers.


MDM: Search > Create
Create 1
PartyId: *
PartyTypeCd: *
Select...
MailingAddressTypeCd: *
DwellingTypeCd:
MailingPreferredCd:
MailingAddressStatusCd:
CountryCd:
TerritoryCd:
CityCd:
PostalCd:
Country_Ext:
United States
State_Ext:
City_Ext:
Alabama
Alaska
Arizona
Arkansas
California
Colorado
Connecticut
Delaware
District of Columbia
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa

# APPENDIX X Dashboard Technology Implementation

### What's In This Appendix

This appendix provides implementation details for dashboard technology in MDM.

Topics include:


- Introduction
- Implementation Steps


# Introduction

The following section describes the dashboard technology implementation steps for creating visual graphs in MDM PGL UIs using D3 (Data-Driven Documents) technology. D3 is a JavaScript library wherein it is possible to create any kind of graph and charts from variety of data sources. In D3 tool, the JavaScript runs inside the browser and makes the rendering faster and smooth transitions from one state to another.

Developing D3 graphs from scratch is time consuming and many plugins like C3 for Line/Bar charts, d3pie for Pie graphs and NVD3 etc. are built on top of D3 for easy implementation of basic graphs without any D3 coding. The PGL developers can relax from third party plugin bugs as these are handled internally inside pglcharts.js.

# Implementation Steps

Perform the following steps to create graphs in MDM PGL UIs:

1 Create any empty PGL container. For example use the below code.

```
<ui:container id="chartContainer1" inner="true"
displayText="LRDM_CODESETS_BY_GROUP" scrollable="true">
    <ui:step id="chartStep1">
              </ui:step>
</ui:container>
```

2 In MDM .pgl files, include pglcharts.js plugin as below:

```
<ui:script language="javascript" type="text/javascript" src="/td/
javascript/pglcharts.js"/>
```

3 Get JSON data in required format.

It can be done via Ajax call or MDM native call.

```
<NATIVE_FUNCTION ClassName="com.teradata.xcore.util.json.XMLJSONUtil"
MethodName="xmlToJSONObjectString" AssignToVar="pieData">
    <PARAM Var="result" Type="xml" />
</NATIVE_FUNCTION>
Sample JSON Data
{"SQL_RESULT":[{"GROUP_ID":"1","GROUP_DESCRIPTION":"Group of OOTB MDM
Code Sets","GROUP_NAME":"MDM Code
Sets","GROUP_COUNT":"22"},{"GROUP_ID":"101","GROUP_NAME":"Group1 Code
Sets","GROUP_COUNT":"2"},{"GROUP_ID":"102","GROUP_NAME":"Group2 Code
Sets","GROUP_COUNT":"2"}]}
```

4 Call pglcharts.js plugin with necessary configuration.

```
pglcharts.generate( {
'charts': [
{
'type':'pie',//Type of graph
'chartStep':'chartStep1',//ID of PGL <ui:step>
'data':{
'src':pieData,//JSON Data
'label': 'GROUP_NAME',//Label key(from JSON data) for Pie grpah
'id': 'GROUP_ID',//Optional miscellaneous data
'value':'GROUP_COUNT'////Value key(from JSON data) for Pie grpah
}
}
]
});
```

## MDM Plugin Features

MDM plugin pglcharts.js features:

- Charts can be organized in the form of rows and columns. Scroll bars are maintained automatically by pglcharts.js plugin for different PGL chart layouts.

  Below is the sample layout with two graphs in one row and one graph in another row.

```
<ui:grid id="chartGrid">
        <ui:row>
          <ui:cell width="50%" valign="top">
           <ui:container id="chartContainer1" displayText="Pie Chart"
scrollable="true">
              <ui:step id="chartStep1" />
            </ui:container>
          </ui:cell>
          <ui:cell width="50%" valign="top">
            <ui:container id="chartContainer2" displayText="Line
Chart" scrollable="true">
              <ui:step id="chartStep2" />
            </ui:container>
          </ui:cell>
        </ui:row>
        <ui:row>
          <ui:cell width="100%" valign="top" colSpan="2">
           <ui:container id="chartContainer3" displayText="Bar Chart"
scrollable="true">
      <ui:step id="chartStep3" />
            </ui:container>
          </ui:cell>
        </ui:row>
```

```
</ui:grid>
```

- Provides an option to render custom graphs as OOTB graphs may not fit to custom requirements.

```
pglcharts.generate( {
'charts': [
{
'type':'pie',
'customChartFunction':renderCutomPie//Custom function name
}
]
});
//Do custom rendering here using any third party plugin, scroll bars
are still maintained by pglcharts.js plugin
function renderCutomPie(){
var chart = c3.generate({
bindto: '#chartStep1_scroller',
size: {
height: 400,
width: 400
},
data: {
columns: [
['data1', 30],
['data2', 120],
],
type : 'pie'
}
});
}
```

- Width and height for graphs is set automatically by pglcharts.js plugin but an option is provided to PGL developers to configure width/height for any custom requirements. minHeight/minWidth can also be set in similar way.

```
pglcharts.generate( {
'charts': [
{
'type':'pie',
'chartStep':'chartStep1',
'width':200,
'height':200,
'data':{
'src':pieData,
'label': 'GROUP_NAME',
'id': 'GROUP_ID',
'value':'GROUP_COUNT'
}
}
]
});
```

- Padding can be used to give space on the sides of graph. Padding is supported only for Bar and Line charts.

```
pglcharts.generate( {
charts: [
{type:'bar',
chartStep:'chartStep2',
padding: {
```

```
top: 40,
right: 100,
bottom: 40,
left: 100,
},
data:{
src:barData,
xAxis: 'name',
yAxis: ['ValuesCount','MappedSourceCount','UnmappedSourceCount'],
onClick:barClickCallback
},
}
]
});
```

- xAxisHeight/xAxisRotate is useful if x-axis tick values are bigger and not visible completely. xAxisHeight/xAxisRotate is supported only for Bar and Line charts.

```
pglcharts.generate( {
charts: [
{
type:'bar',
chartStep:'chartStep2',
xAxisHeight:200,
xAxisRotate:60,
data:{
src:barData,
xAxis: 'name',
yAxis: ['ValuesCount','MappedSourceCount','UnmappedSourceCount'],
onClick:barClickCallback
},
}
]
});
```

- To maintain correct scroll bars for different layouts, a configuration option 'align' is given to adjust height of two graph containers. This is especially useful when graphs are arranged side by side using <ui:cell>

```
pglcharts.generate( {
'charts': [
{
'type':'pie',
'chartStep':'chartStep1',
'align':{
'height':'chartStep2'
},
'data':{
'src':pieData,
'label': 'GROUP_NAME',
'id': 'GROUP_ID',
'value':'GROUP_COUNT'
}
},
{
'type':'line',
'chartStep':'chartStep2',
'data':{
'src':lineData,
'xAxis': 'name',
```

```
'yAxis': ['ValuesCount','MappedSourceCount','UnmappedSourceCount']
}
},
{
'type':'bar',
'chartStep':'chartStep3',
'data':{
'src':barData,
'xAxis': 'name',
'yAxis': ['ValuesCount','MappedSourceCount','UnmappedSourceCount']
}
}
]
});
```

- Drill down on click of graph

```
pglcharts.generate( {
charts: [
{
type:'bar',
chartStep:'chartStep2',
data:{
src:barData,
xAxis: 'name',
yAxis: ['ValuesCount','MappedSourceCount','UnmappedSourceCount'],
onClick:barClickCallback
}
}
]
});

function barClickCallback(c,d, element) {
var codeSetName = c.categories()[d.index];
document.form.FILTER_Code_Name.value=codeSetName;
document.form.BUTTON_ID.value = 'TABULAR_VIEW';
document.form.submit();
grayOut(true);
}
```

## Sample PGL Files

MDM provides the following sample pgl chart files.

Figure 112: Sample PGL Chart Files

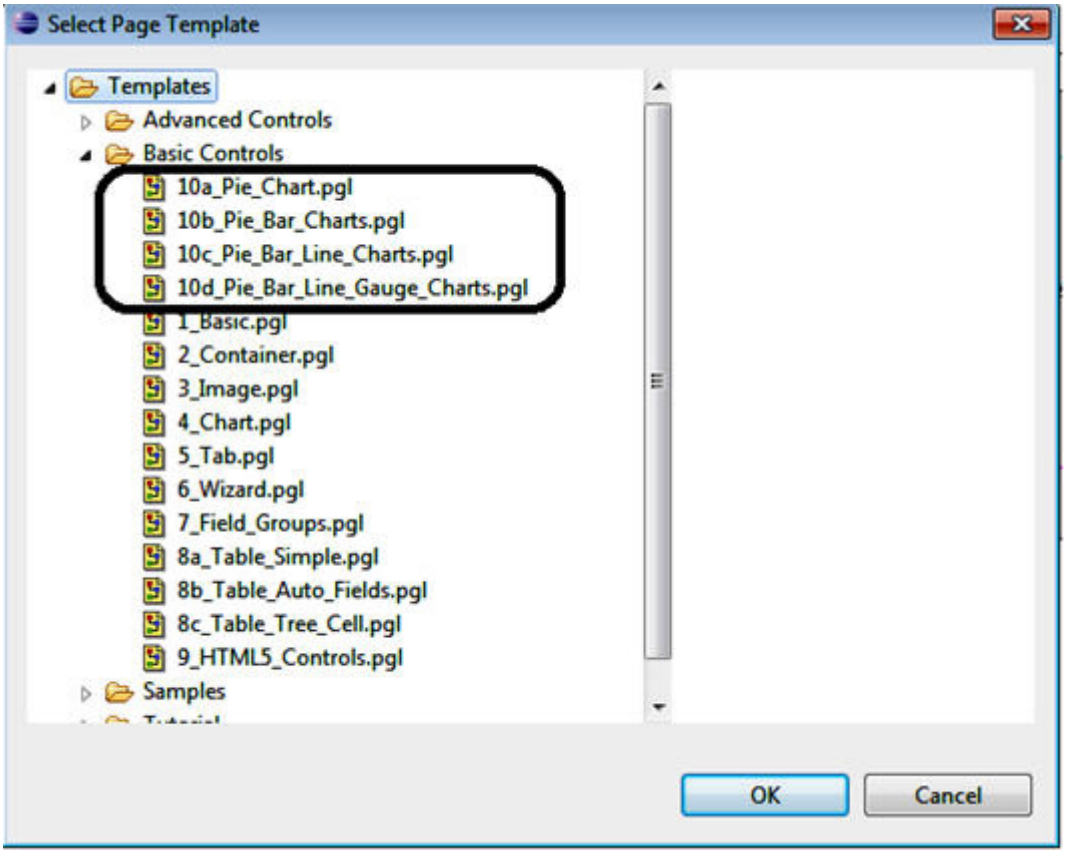



Figure 113: Sample Pie_Chart

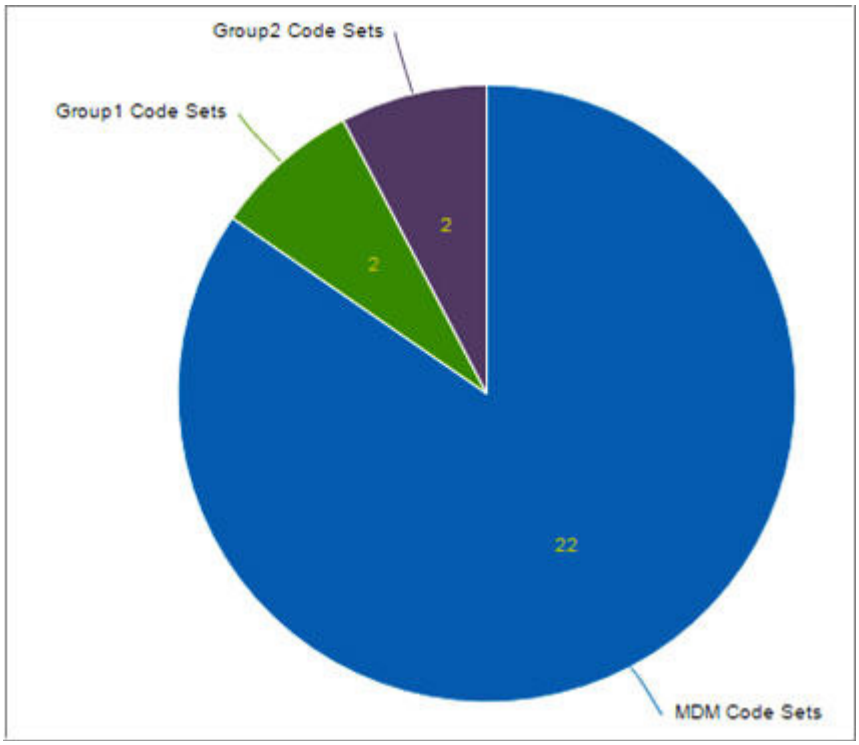


Figure 114: Sample Pie_Bar_Chart

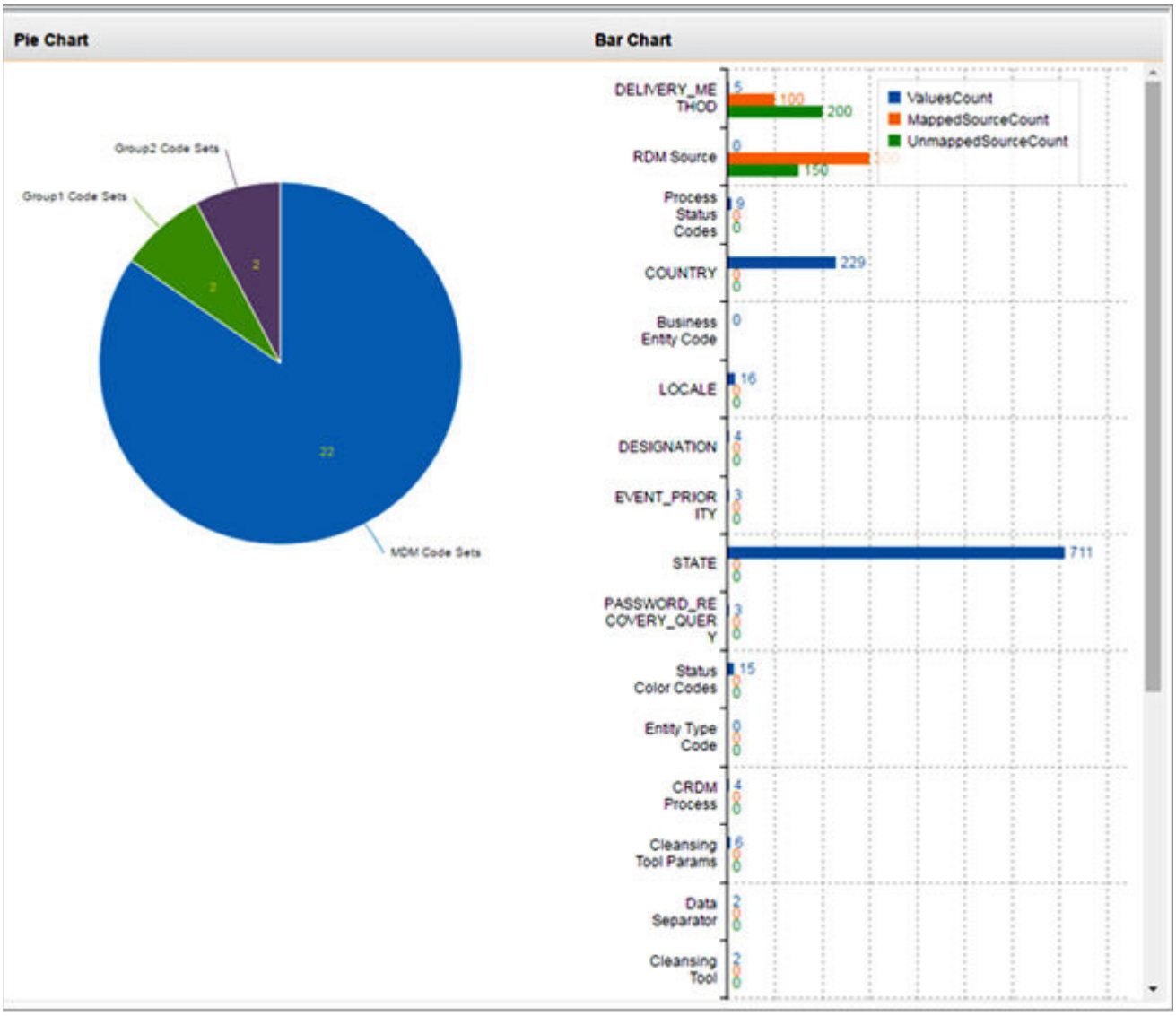


Figure 115: Sample Pie_Line_Bar_Chart

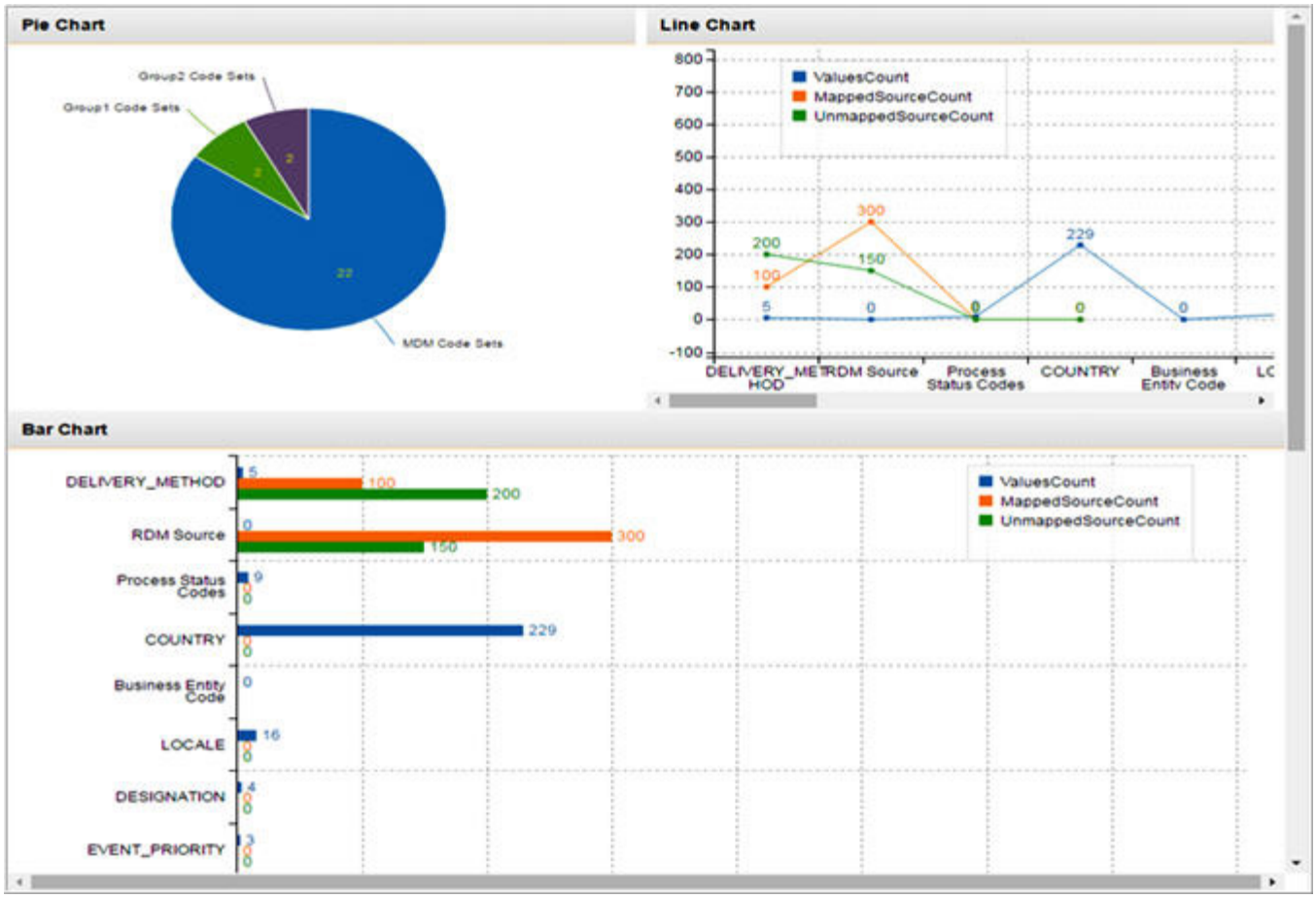
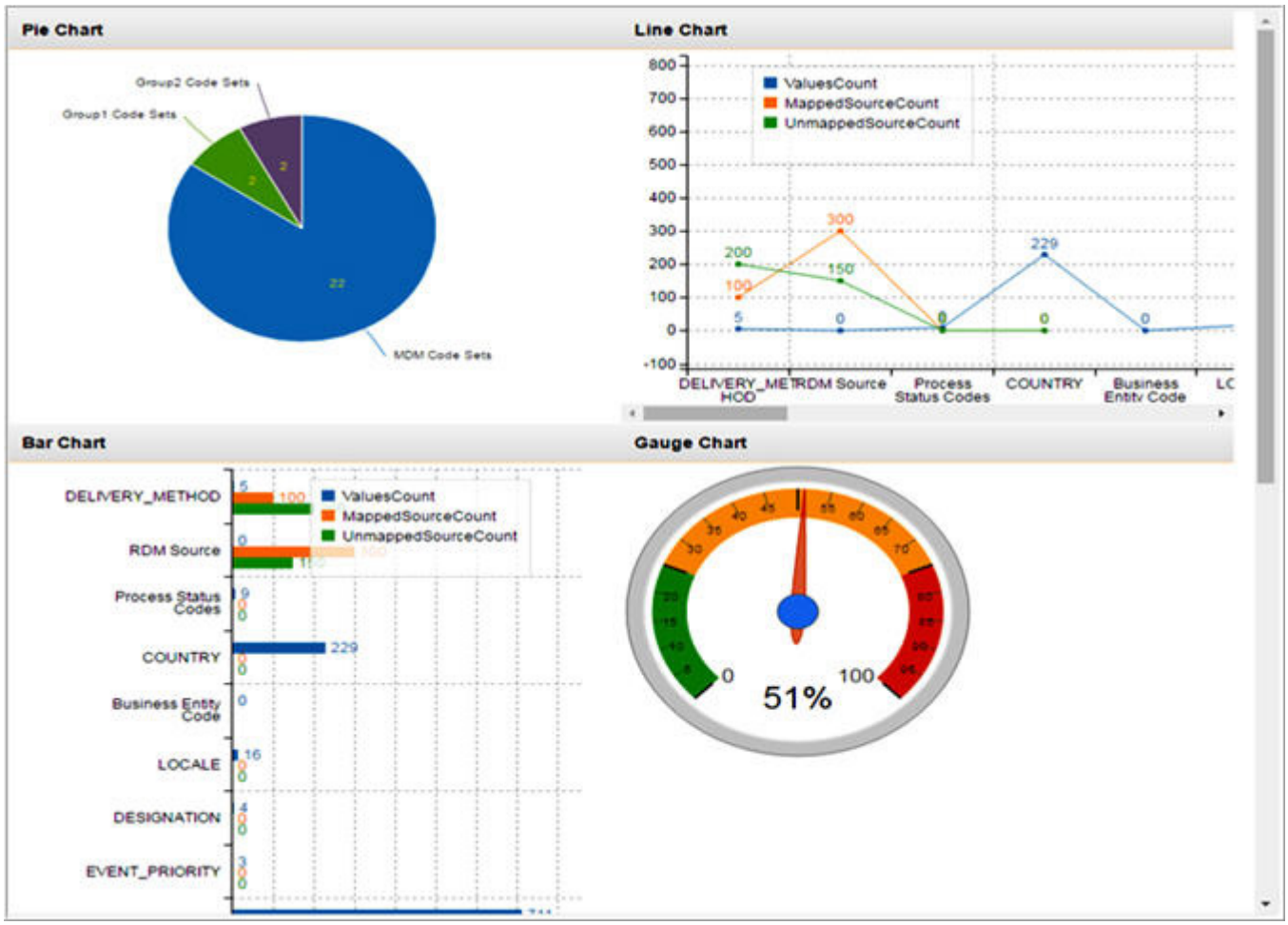


Figure 116: Sample Pie_Line_Bar_Gauge_Chart



# APPENDIX Y Angular UI for Custom Workflows

### What's In This Appendix

This appendix provides implementation details for adding custom services at runtime.

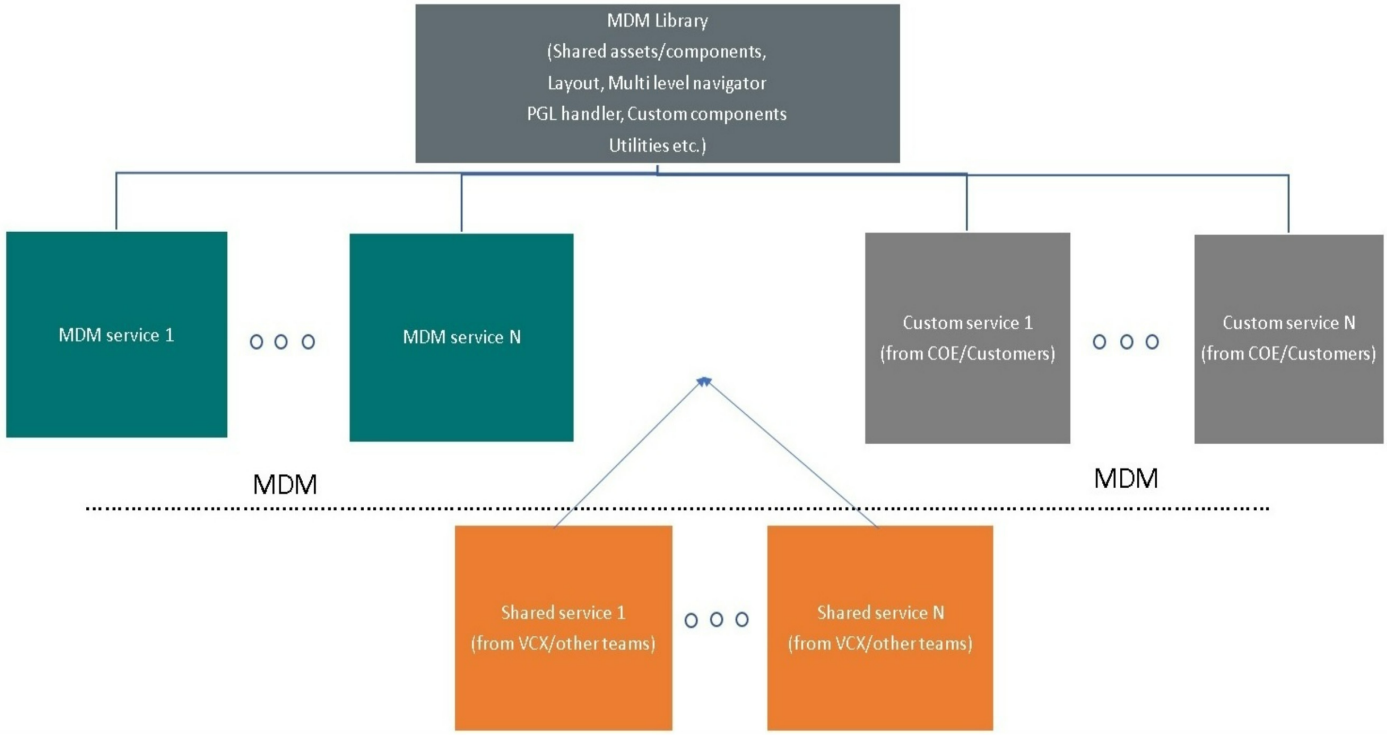
Topics include:


- Introduction
- Implementation Process


## Introduction

Angular UIs for custom workflows can be built and deployed at runtime on MDM server without having access to OOB angular UI code. Figure 117 displays the Angular UI architecture. This section explains how to add custom services at runtime.

Figure 117: Angular UI Architecture



# Implementation Process

Integrating Angular custom application to MDM, require following steps:

1. Get a Sample Project and Check for Various Configuration
2. Add Custom Services into Sample Project
3. Production Build of Custom Project
4. Deploy Custom Application in MDM
5. Create Left Navigator Link
6. Access Custom Application from Left Navigator

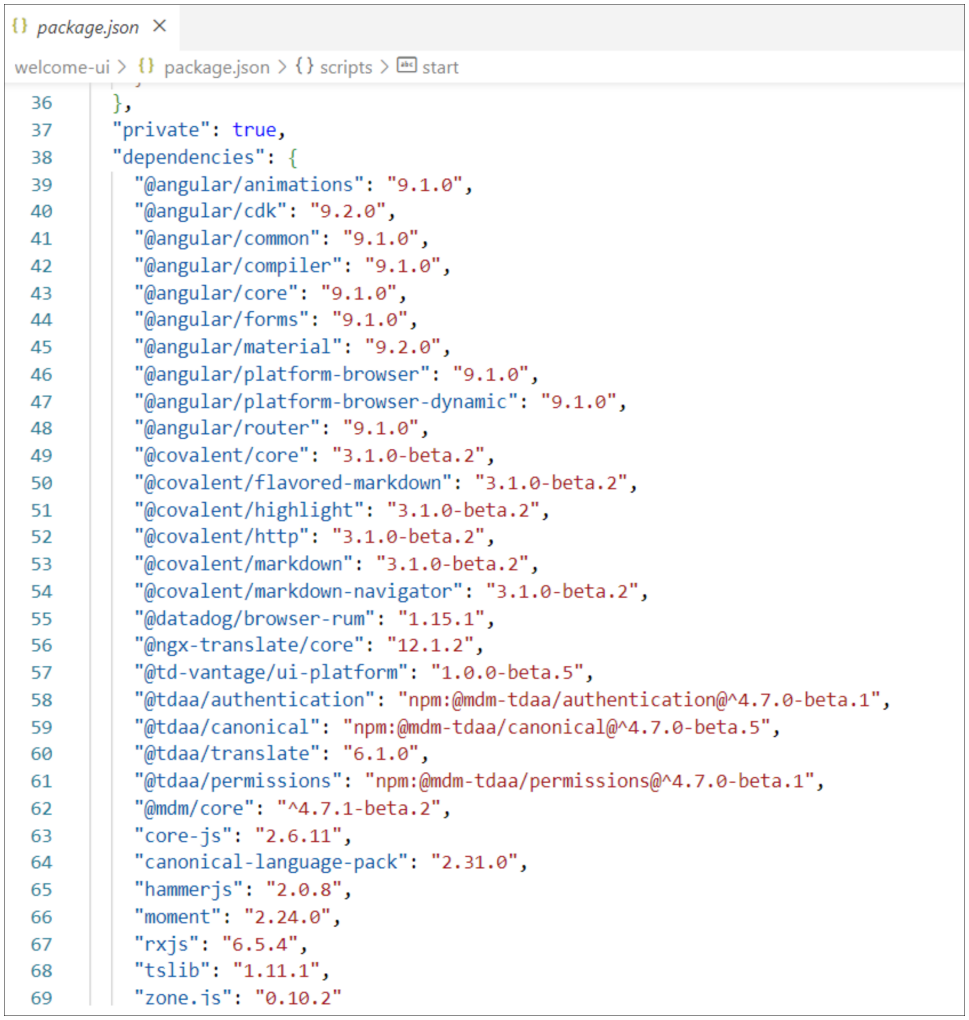
## Get a Sample Project and Check for Various Configuration

1. Download the welcome-ui from https://raleigh-git.td.teradata.com/MDM/services/welcome/welcome-ui as sample project.

   **Note:** You can connect with the engineering team to obtain the details of latest sample project.

   The downloaded sample project comes pre-configiured with Angular, Covalent and any Teradata internal(e.g. TDAA) dependencies.

   PACKAGE.JSON provides the list of libraries used in the project.




2 You can use any Covalent, TDAA components in custom application. For details, see https://teradata.github.io/covalent

3 To view the list of components, download and run TDAA library (https://raleigh-git.td.teradata.com/tdaa-ui/tdaa-ui).

**Note:** package.json contains convenient scripts, you can use these scripts through Yarn.

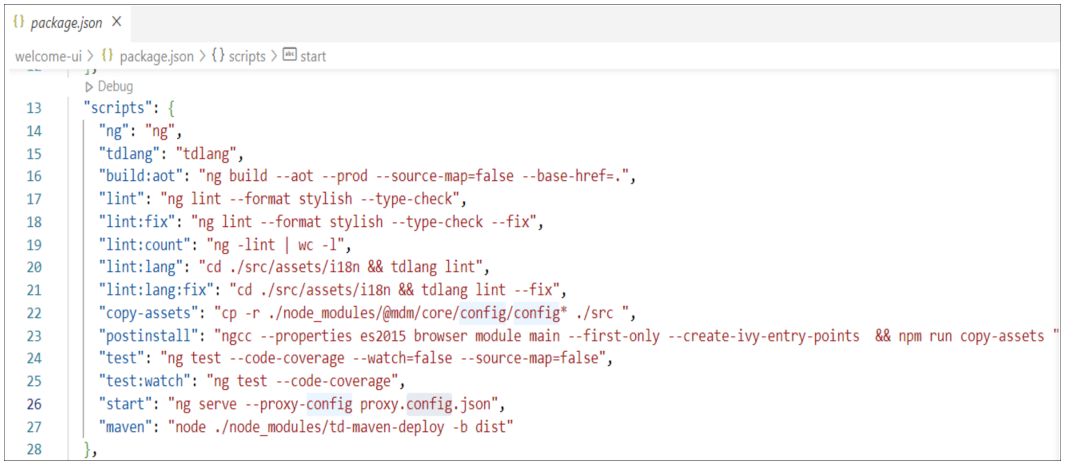
yarn postinstall → run post install scripts. e.g. copy files from node_modules to custom project folder for easy access

yarn start → start Angular CLI server with proxy

yarn lint → shows any code styling errors

yarn build:aot → generate production build content

yarn test → run unit test cases




4 To connect the sample project with other Angular services for sharing or accessing data (for example, authentication, metadata services), configure the proxy.config.json.

**Note:** You can configure the proxy.config.json only in test environment and not in production environment.

In below configuration 'target' URL is the server where other services are running. Any REST call URL fragments, from sample project, containing 'api' or 'metadata' will redirect to respective 'target' URL to get/share data

5 To view the global style sheets, see angular.json

6 Files are listed in order of priority - low to high i.e. theme.scss is highest priority in below configuration and it will override any styles in other files.

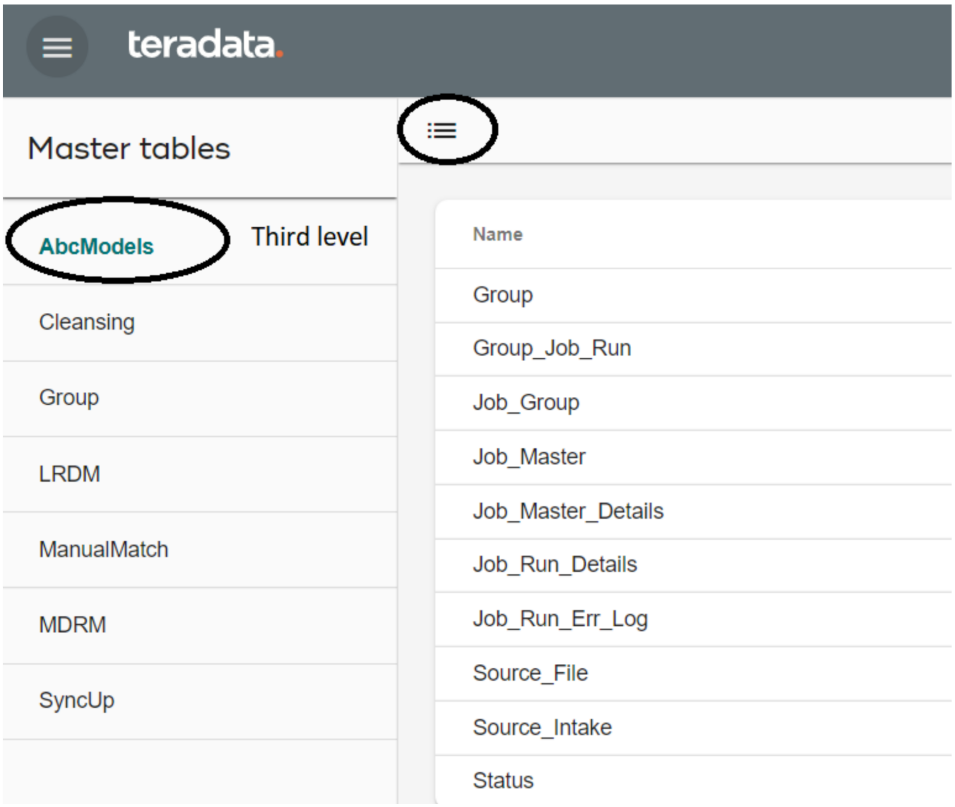
7 Custom global CSS files can be created and add entry here to take that as priority.

8 Left navigation config (config.js file)

Config.js file includes left navigator configuration and Angular development environment configurations.

This file will be replaced with config.prod.js file in production environment. Set the following parameters:

- LEFT_NAV_THIRD_LEVEL_COLLAPSIBLE: set to true if want to collapse third level on click of  link in fourth level. New menu icon appears to expand it again.
- LEFT_NAV_FIRST_LINK_AUTO_RENDER: set to true if want to render first link (of third level) in center panel, on open of third level in left navigator.




9 Permission Configuration (permissions.config.ts)

To configure permissions of Angular route, you can use permissions.config.ts file.

You can access route only if activity assigned in 'User Management' modules. You must configure Activity name in angular also.

For example, if activity 'crossReferenceActivity' assigned to user in User Management, then add same in below Angular file and then mention in route file.

permissions.config.ts

cross-reference-ui > src > config > permissions.config.ts > ...

```
15
16  // NOTE: Add permission key definitions here...
17  export enum PermissionTypes {
18    MANAGE_CROSS_REFERENCE = 'crossReferenceActivity',
19  }
20
21  // NOTE: A nested permission structure allows us to define our permissions
22  //       around feature sets.  We can also group permission types for allowing
23  //       access to a feature if a user has any of the following types...
24
25  //       For now try to use the following for permissions mapping in your applicat
26  //       instead of pulling a type directly
27  export const PERMISSIONS: any = {
28    CROSS_REFERENCE: {
29      // NOTE: if a user has any of these permissions, they should probably be able
30      //       see the index page...
31      INDEX: [PermissionTypes.MANAGE_CROSS_REFERENCE],
32    },
33  };
```

Update route files as below:

app-routing.module.ts

ss-reference-ui > src > app > app-routing.module.ts > routes > children

```
3  const routes: Routes = [
4    {
5      path: '',
6      component: MainComponent,
7      children: [
8        {
9          path: 'entity-list',
10         loadChildren: () => import('./cross-reference/manage-cross-reference.module').then(m => m.ManageCrossReferenceModul
11         canActivate: [TdaaAuthenticationGuard, MdmPermissionGuard],
12         data: {
13           permissions: PERMISSIONS.CROSS_REFERENCE.INDEX,
14         },
15       },
```

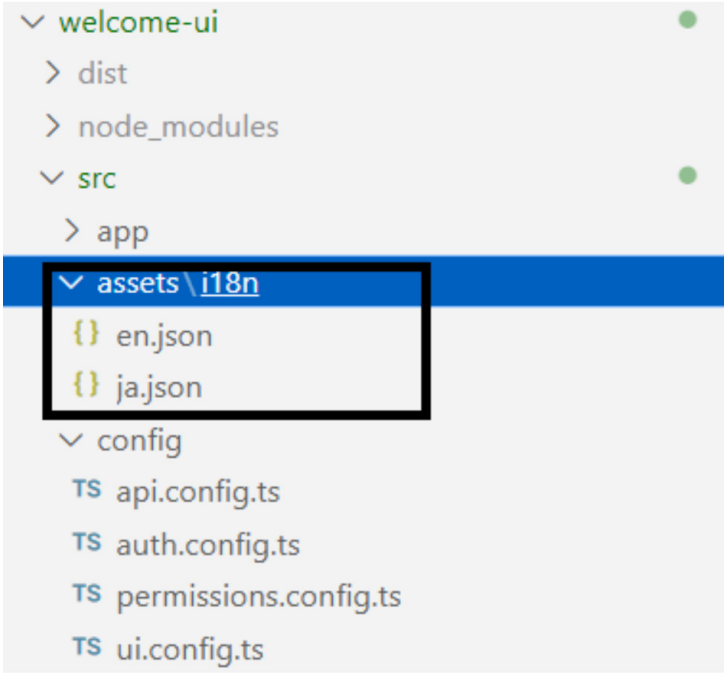
To provide permission to all users, remove 'MDMPermissionGuard' in above code.

**10** Country specific Files:

i18n: Add country specific i18 files under src/assets/i18n

For i18 to work, ensure that locale of browser has changed under browser settings. Also, locale in MDM User Profile UI should match with browser locale.

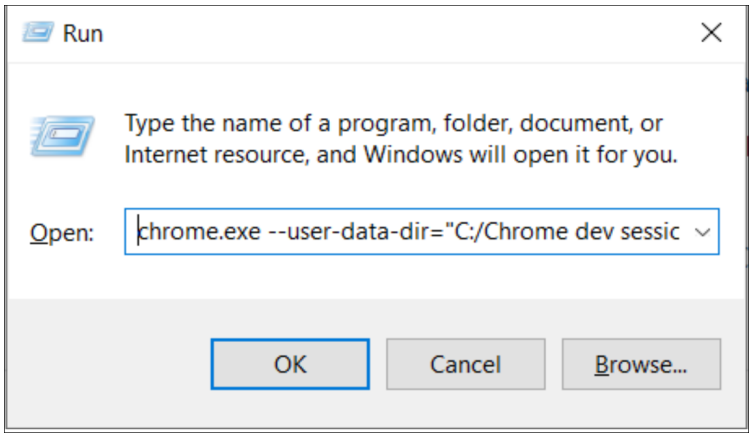
Once these changes are done then clear browser cache and refresh UI to see i18 changes




## Add Custom Services into Sample Project

Before adding any custom application code to sample project, you must execute the sample project without any errors. Follow the mentioned steps:

1. Install dependencies in sample project using 'yarn install'
2. Execute 'yarn start' and fix any errors.
3. Open Chrome using 'chrome.exe --user-data-dir="C:/Chrome dev session" --disable-web-security'



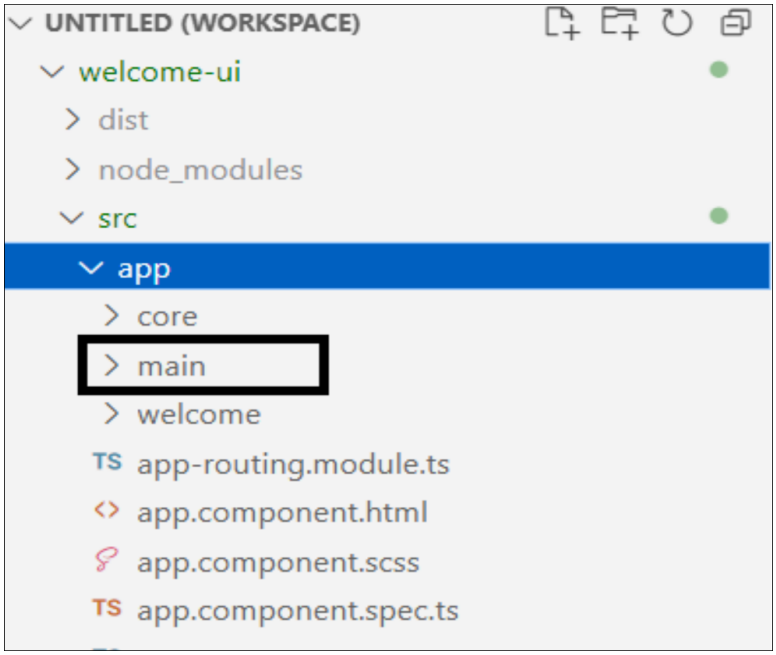

4. Logon MDM.
5. In the same browser, open new tab and access http://localhost:4200.

   Ensure that the sample project loaded with left navigator and toolbar without any issues.
6. Search for 'welcome' keyword in sample project and replace with custom project name.

7 Add custom specific Angular code under 'src/app'. No changes required in files under 'main'

8 'welcome' and 'core' folders contain welcome project specific code. Add new folders following similar structure and then remove 'welcome' and 'core'

9 Add as many folders/files based on custom project requirement. Also add necessary routes, styles, i18, permissions etc. similar to sample project




REST API Calls

Angular REST calls are usually made from Angular service files. An example for service file can be seen below.

```
getCrossRefObjList(): Observable<any> {
    const body = {
    };
    const BASE_URL = `${this.sharedService.getAppServerUrl()}${this.sharedService.getContextPath()}/api/v1`;

    return this.http.post(`${BASE_URL}/services/MDMServices/rules/xref-entities`,
        body, {
        headers:
        {
            Accept: this.appConfig.accept,
            response: this.appConfig.response,
            Authorization: this.appConfig.authorization,
            'return-response-array': 'true',
        },
        observe: 'response',
    }).
        pipe(
            map((res: HttpResponse<any>) => {
                return res.body;
            }),
            catchError((err: HttpErrorResponse) => {
                // Handle Http Errors
                // FIXME: REPLACE WITH [] WHEN API WORKS
                return of([]);
            }),
        );
}
```

You can use BASE_URL and headers in same way. In the below REST API, 'xref-entities' is the PGL XRULE name. Pass any input data to XRULE using 'body'.

Response JSON is available in 'res' variable.

## Production Build of Custom Project

Ensure that the custom application is working as expected by accessing http://localhost:4200

Execute below commands before production build (optional):

yarn lint:fix

yarn lint:lang:fix

yarn test

To generate production build content under <PROJECT_ROOT>\dist, use command 'yarn build:aot'

## Deploy Custom Application in MDM

Take production build content from <PROJECT_ROOT>\dist and place it under below location

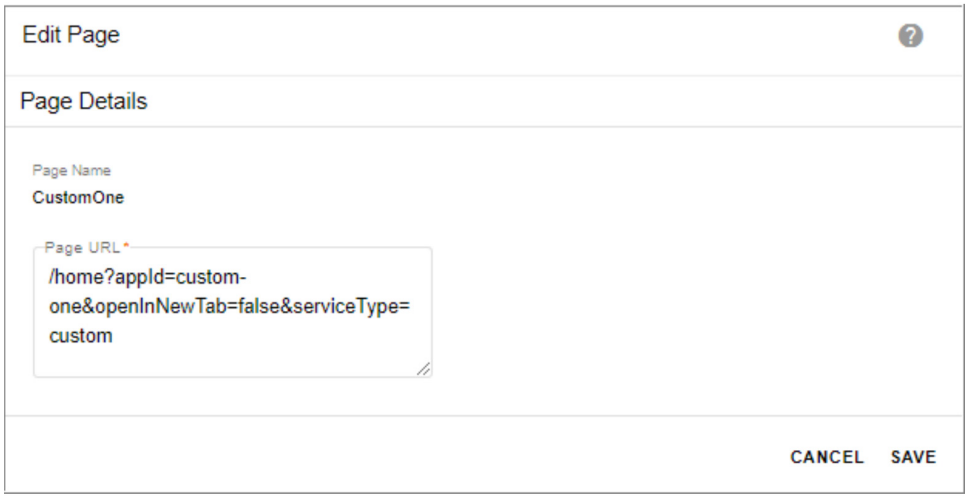
<MDM_Iinstall_Directory>\web\mdmclient\covalent\<CUSTOM_FOLDER_NAME>

CUSTOM_FOLDER_NAME can be any suitable name for custom application. Same name is used in MDM menu builder to create left navigator link.

## Create Left Navigator Link

Navigate to MDM Menu Builder → Manage pages.

In below UI, '/home' is Angular route.This route should match with route in Angular project




For empty routes mention '/'.

You can mention the url for empty routes as below:

'/?appId=custom-one&openInNewTab=false&serviceType=custom'

The empty route url has following parameters:

- appId: can be any name. This should match with the folder name created under \covalent.
- openInNewTab: true/false

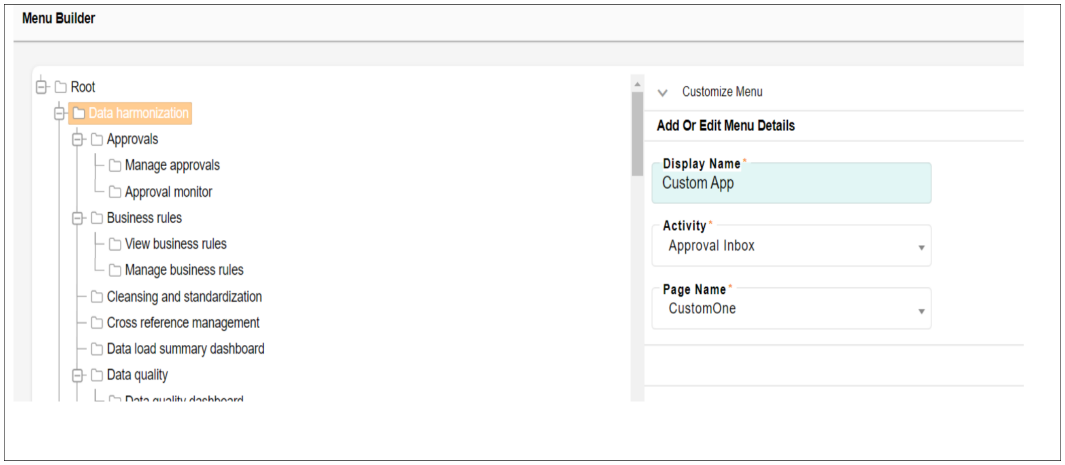
  Mention 'true' if want service to be opened in new tab. Mention 'false' if service to be loaded in same browser window/tab. This is deprecated from 4.7 onwards.
- serviceType: mdm/custom/shared

  -All OOB services are of type ‘mdm’

  -All custom services are of type ‘custom’

  -All shared services from other teams like VCX, are of type ‘shared’

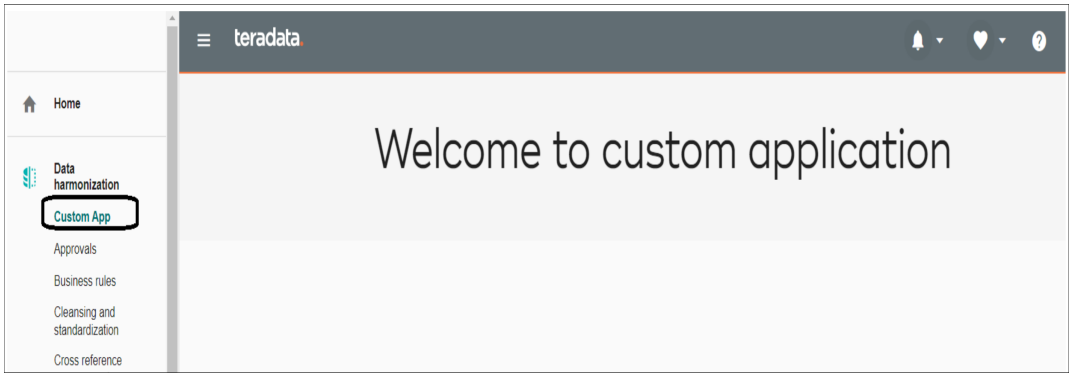
  Create left navigator link using above page.




## Access Custom Application from Left Navigator

Access custom application link from left navigator.

**Note:** You need not to restart servers. Clear browser cache and reload browser URL.



# APPENDIX Z Error Handling for Unavailable Shared Services

### What's In This Appendix

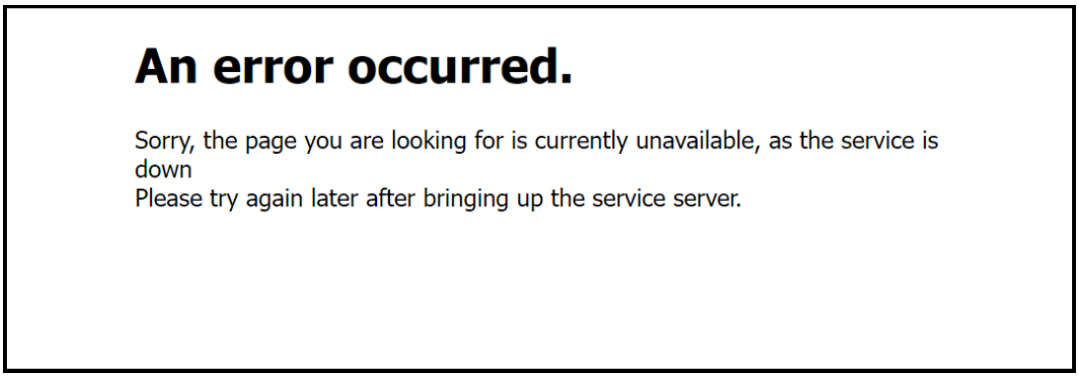
This appendix provides implementation details for error handling of Left navigation click when shared services are unavailable in MDM.

Topics include:


- Introduction
- Implementation Steps


## Introduction

The following section describes the error handling steps implementated for shared services when they are unavailable. When you do not install shared services while installing MDM and try to access shared services like Data Harmonization, Cleansing and Standardization, and others, MDM shows an error for unavailability of the services.




When a service is unavailable, MDM returns 502/503 as the corresponding HTTP response code.

## Implementation Steps

You can find nginx default configuration for the codes 502/503 in the nginx.conf property file under 'nginx\conf' folder.

Therefore, by default it reroute to 50x.html upon receiving one of these codes. You can find the html file under ‘nginx\html’ folder.

To change the display message, when you click the left navigation while any service is unavailable, you can change the content inside this html file.

To display any other html page, configure the same in nginx.conf property file.

Sample Example:

```
<!DOCTYPE html>
<html>
<head>
<title>Error</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>An error occurred.</h1>
<p>Sorry, the page you are looking for is currently unavailable, as the service is down<br/>
Please try again later after bringing up the service server.</p>
</body>
</html>
```

# APPENDIX AA Frequently Asked Questions

### What's In This Appendix

This appendix provides solutions to frequently asked questions.

## FAQs

***Q 1:*** *I cannot see my table displayed properly. It shows up like in the screenshot below. Why?*

**A.** Set the scrollable attribute on the table to "yes"

Example

```
<ui:table …. scrollable="yes"/>
```

***Q 2:*** *I have multiple tables in one page. How do I make pagination on the table work?*

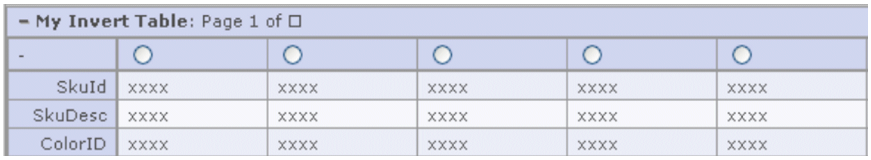
**A.** Use the `START_COUNT` for each table which will be returned in a parameter `<tableId_START_COUNT/>`

Example

If you have two tables `bottomTable2` and `bottomTable3`, thisParam will have the following.

```
<PARAMETERS>
   <bottomTable2_START_COUNT Value="0"/>
   <bottomTable3_START_COUNT Value="0"/>
</PARAMETERS>
```

***Q 3:*** *I cannot see the pagination information on the table properly as shown in the figure below. Why?*




**A.** Set maxRows attribute for the table.

Example

```
<ui:table …… maxRows=”-1”/>
```

***Q 4:*** *How do I override the default behavior of a button?*

**A.** Add a javascript function with the same name as the one called for button action at the end of the PGL.

Example

If you have a button called Search, the function would be `on_Search()`. You can confirm this by doing a View Source on the page (right click). You can copy the default javascript function and add overrides to it at the end of the PGL.

Add the following code at the end of the PGL.

```
<ui:script>
   on_Search(){}
</ui:script>
```

***Q 5:*** *How do I Internationalize display text in MDM UI workflows?*

**A.** Solution:

1. Create the resource bundle i18n-messages_LANG.properties for the specific language. It will have contents such as the following:

   ```
   Field\ Name=(encoded string in UTF-8)
   ```

2. In the PGL

   ```
   <ui:display-field id="FIELD" name="FIELD" displayText="Field
   Name" required="true"/>
   ```

3. Put the i18n-messages_LANG.properties in `STUDIO_HOME/services/ui/WEB_INF/classes` (LANG can be fr,en_US,zh_TW etc)

***Q 6:*** *I see this error on WebLogic console. How can I fix it?*

```
java.net.ConnectException: Connection refused: connect
at java.net.PlainSocketImpl.socketConnect(Native Method)
at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:305)
```

**A.** Make sure your locator service is running.

***Q 7:*** *I am getting a* `NullPointerException` *on the WebLogic console and UI node in studio. How do I debug this problem?*

**A.** Check if the PGL is valid (missing id attributes.)

***Q 8:*** *How do I set values dynamically in PGL?*

**A.** Solution(s):

1. Set variable in pre processing

2. Access in PGL

Example

```
<SET Var=”fieldValue” FromValue=”test”/>
<ui:entry-field name=’EntryField” value=’$root/fieldValue/
@Value’/>
```

***Q 9:*** *How can I hide a field at runtime based on a condition?*

**A.** Use `displayCondition` on the component

Example

```
<ui:image src="{$root/Image/@Value}" displayText="Attached Image"
displayCondition="{string-length($root/Image/@Value) &gt; 0}"
height="150" width="150" />
```

***Q 10:*** *Can I set the width for an empty list box ?*

**A.** The only way to control the width for empty listbox is using javascript. Call this javacsript function in your page `onLoad()` function.

***Q 11:*** *Can I have an image inside a button?*

**A.** It is not possible to embed image within a button.

***Q 12:*** *I have a screen, where form needs to be self-submitted on change of a drop-down. Is there any way to map submit action with transitions so that I can submit to different pages as per conditions?*

**A.** Use `onChange` handler in ui:dropdown element for invoking a javascript function where you can submit the form with your required action (`BUTTON_ID`)

***Q 13:*** *I want Export To Excel functionality for a table. How is the functionality implemented in PGL?*

**A.** Set attribute `exportable="true"` in `ui:table`. You will find an icon in the top right corner of the table on the UI.

Example

```
<ui:table id= ….. exportable=”true”/>
```

***Q 14:*** *Do I have to specify both name and id attributes on all the components?*

**A.** For all the pgl elements, specifying 'name' attribute is enough. The 'id' attribute is a must only for ui:button.

***Q 15:*** *What should I do for changes in the UI workflows to reflect on the page? Should I restart WebLogic?*

**A.** Whenever you change your workflow or a rule in Studio, you do not need to restart WebLogic. You can either restart your Local Server or just reload the form (use the icon to the left of Start Local Server).

You can just hit the Refresh button on your browser to update the Tasks list

***Q 16:*** *What should I do for pgl changes to reflect on the page? Should I reload the local server?*

**A.** Hit the apply button on the pgl editor. You do not have to reload the local server.

***Q 17:*** *How do I retain the selections in a paginated table across pages?*

**A.** Set "`rememberSelections`" attribute to "`true`" on the table

Example

```
<ui:table …. rememberSelections=”true”/>
```

***Q 18:*** *How can I pass a parameter to a workflow from a link-field ?*

**A.** Solution(s)

1. Pass the parameter value using select in the link field. (Give the value depending on how the parameter is set in your $root in pre-action)
2. Capture the parameter in $thisParam of the start node of the workflow you are passing it to.

Example

If you want to pass ItemID from the following data into the workflow ItemPopUp

1. Data in Pre-Action

```
<SAMPLE_XML>
  <ItemID Value=”01”/>
</SAMPLE_XML>
```

2. PGL

```
<ui:link-field id="ItemID" name="ItemID" displayText="ItemID"
targetType="popup" targetName="catalogPopup">
<ui:url page="/base/start.x2ps">
   <START_WORKFLOW select="'ItemPopUp'"/>
   <myParam select="ItemID/@Value"/>
</ui:url>
</ui:link-field>
```

3. Start Node of the ItemPopUp workflow

```
<SET Var="ID" FromValue="{$thisParam/ItemID/@Value}"/>
```

***Q 19:*** *How do I refresh my page based on a selection in a dropdown.*

**A.** Specify onChange attribute with a javascript that will have the action to refresh the page.

Example

1. Data

```
<ROOT>
<CITY>
  <City1 Option="one" Value="one"/>
  <City2 Option="two" Value="two"/>
</CITY>
</ROOT>
```

2. PGL

```
<ui:dropdown id="CITY" name="CITY" displayText="City"
optionSrc="$root/CITY/*" onChange="Javascript:onLevelChange()"/>
```

3. Script

```
<ui:script>
function onLevelChange()
{
  document.forms[1].BUTTON_ID.value =
  &quot;LEVEL_PRESSED&quot;;
  document.forms[1].action = 'view.x2ps';
  document.forms[1].submit();
}
</ui:script>
```

***Q 20:*** *How can I create a self loop?*

**A.** Place your mouse pointer on the "source" node. You will see a Red-arrow-in-a-red-circle. Click it, drag and drop onto the "target" node.

To create a link onto the node itself, press the shift-key before dropping the link back onto the node.

To change the shape of the link, click anywhere on the link (it will change color to red), drag it while keeping the shift-key pressed

***Q 21:*** *I am not able to see any links in the left navigation bar. I am also getting a* `UI_SESSION_EXPIRED` *error on the server. How can I fix this problem?*

**A.** Solution(s):

1. Check if your locator port is consistent across

   Xserver.xml, ui_in_process.xml, xcoreweb.properties, locator.xml, bootstrapXserver.xml

2. Ensure that the supported protocols dialog in Xserver configuration lists only http (Remove xrmp and tcpa and restart local server)
3. Check if weblogic server is calling the getNavigation method

***Q 22:*** *How can I define global variables specific to particular workflow in PGL?*

**A.** Use pipeline variables in the UI workflow to store global variables.

***Q 23:*** *How can I define global variables specific to particular service in PGL?*

Use scope attribute as shown below.

```
<SET Var="globalValue" FromValue="99" Scope="Global"/>
```

***Q 24:*** *How do I invoke an existing xservice from a ui workflow that is being built in studio?*

**A.** Check if the rule is public:

Invoke rule from the workflow

```
<REQUEST Name="ruleName" ServiceName="ServiceA">
  <param...
</REQUEST>
```

***Q 25:*** *Can I plug in an existing UI into studio UI workflow?*

**A.** UI workflow can invoke existing ui using javascripts.

***Q 26:*** *I am getting the following error in the workflow*

```
ERROR READS: PLEASE CONTACT ADMINISTRATOR, CAUSE:PROBLEM
IN POST-ACTION: CONNECTION RESET BY PEER,EXCEPTION IN XRMP CAN
HANDLE
```

How can I avoid it?

**A.** Your xserver.xml should have only `http` in the protocols list. Remove `tcpa` and `xrmp` if you have them

***Q 27:*** *I have a multi tabbed container. How can I set data for the active tab in the workflow?*

**A.** Solution

1. Define Step id for each tab in the PGL
2. Get the selected tab id from the $thisParam variable. The parameter to look for is `{container_id}_STEP_ID`

Example

PGL

1. `<ui:container id="topContainer">`

```
<ui:step id="tab2" displayText="Tab 2">
```

2. `<SET Var="tabId" FromValue="{$thisParam/ topContainer _tab2/ @Value}"/>`

Can I change `UI_WORKFLOW` and call it something else?

Change the following in `STUDIO_HOME\services\ui\WEB-INF\tutorial\model\core\navigation`

1. activities.cnd

```
<xapl:variable name="studioNavigation">
  <xcore:execute-method name="//UIWorkflowService/
  getNavigation" entireResponse="false" />
</xapl:variable>
```

2.

```
<x2:command public="yes" description="_" name="init">
  <xapl:variable name="SelectedActivity"
  resolver="session" global="true"/>
  <xcore:define-service name="UIWorkflowService"
  service="UI_WORKFLOW"/>
</x2:command>
```

***Q 28:*** *I am getting this error on the UI. How do I get around it?*

```
Please contact administrator
Page group template null is not loaded into cache Index: 0 Size: 0
```

**A.** Ensure that your pgl is valid (If you copied it from some other editor, there could be escape characters in the pgl) and is in the same directory as the workflows.

***Q 29:*** *I am getting this error in the left navigation on the UI. What could be the reason?*

| | |
|---|---|
| Please contact administrator | |
| | Page group template null is not loaded into cache Index: 0 Size: 0 |

**A.** Ensure that your services are up and registered with the locator

***Q 30:*** *I am seeing the following on the UI. What could the reason?*

**A.** Check if your server is up.

***Q 31:*** *I am seeing the following error at startup of local server. What could be the reason?*

```
<RESPONSE Status="Error" Description="error creating DebugProxy:
java.rmi.server.ExportException: Port already in use: 1099; nested
exception is:
java.net.BindException: Address already in use: JVM_Bind"/>
```

Go to **Tools** | **Advanced** in Studio. Change the debug port from 1099.

***Q 32:*** *I used link to popup a window to select data for an entry-field in the main window. How do I return the selected data to the entry-filed in the main window? and how do I close the popup window?*

**A.** Override the on_select (assuming your button id is select) javascript function in your search pop up pgl as shown below.

1. Get the selected value
2. Update the opener window's entry field with the selected value
3. Close the pop up

Example

```
<ui:script>
    function on_select()
    {
      var selectedValue = document.form.SELECTED_ID.value;
      parent.opener.form.<entry-field>.value = selectedValue;
```

```
        window.close();
      }
</ui:script >
```

***Q 33:*** *Where can I find the automatically generated java script for buttons?*

You can get this javascript function by doing 'view source' on your displayed page. The auto generated javascript name is always on_button_name. For example if your button name is "close" then your javascript function would be "`on_close ()`".

***Q 34:*** *How can I delete "denotes required field" Message?*

**A.** `<ui:validation displayCondition="false"/>`

***Q 35:*** *How do I make a table column sortable?*

Set `sortOrder="Ascending"` on the table and `sortable=true` on the table column. You will get your selection as `'SORT_BY'` and `'SORT_ORDER'` in `$thisParam` variable. You can use these parameters to sort the data which is being fed to the table.

***Q 36:*** *Can Tooltip functionality in PGL UI display server-side information?*

**A.** This feature is supported for ui:image-link-field or ui:link with image in ui:header. For ui:link field with image, the displayText value of the field is displayed as a Tooltip. For ui:image-link-field the attribute alt contains the value so that OnHovering the mouse value of this attribute is displayed as a Tooltip. Example:

**1** ui:link

```
<ui:header>
<ui:link displayText="{$root/pagevalue/@Value}">
<ui:url page="{$pages:note_details}"/>
<ui:image src="/view_details.gif"/>
</ui:link>
</ui:header>
```

**2** ui:image-link-field

```
<ui:image-link-field displayText="Impact Analysis" src="/
view_details.gif" name="ImpactAnalysis" dataType="Constant"
alt="{$root/pagevalue/@Value}">
<ui:url page="{$pages:ImpactAnalysis}">
<ID select="ID/@Value"/>
</ui:url>
</ui:image-link-field>
```

***Q 37:*** *How can I preserve leading or trailing whitespace in the xml element? For example,* `<ADR_NOTES> Starbucks of Beaverton Oregon Starbucks of Beaver</ADR_NOTES>`

**A.** Add reserved xml attribute, xml:space="preserve" to the element or its parent. For example, `<ADR_NOTES xml:space="preserve"> Starbucks of Beaverton Oregon Starbucks of Beaver</ADR_NOTES>`

***Q 38:*** *In the Enterprise Model, for User Groups there is an organization specification - What do the System vs Default picklist choices mean?*

**A.** It does not matter which one you choose out of the box as that is just to show-case that customers may have their own organizations/departments created within MDM for user groups segregation. It is used in PIM.

***Q 39:*** *In the Enterprise Model, for User Groups there is a Type specification - What do the Administrator/Enterprise/Partner picklist choices mean?*

**A.** This classification is used to segregate MDM users of the group Administrator or Enterprise or Partners. These are the out of the box group types provided but users/ customers can always create their own required groups to manage roles and users. That is, Administrator group user will have admin rights of MDM, Enterprise grp users will have MDM features access while Partners may have access to data upload etc.

***Q 40:*** *In User Activities, there is a User Activity Type specification - What do the UIForm, MenuForm, Admin picklist choices mean?*

**A.** UIForm type belongs to Table Editor (old x2) activities which is internal to MDM although user can create an activity as UIForm.

MenuForm type is the one which is generally used while defining activities to show-up URLs on the left navigation screen.

Admin type belongs to the MDM user mdm_admin which is a dummy admin user to have only Admin activity role.

***Q 41:*** *In what circumstances do we actually "insert" a new service?*

**A.** “Customize Service” option is basically to have child MDM service to the Base MDM Master service. (Most of the deployments does this for their subject area that is, CDI, PIM etc.)

While Insert Service will be useful to add infrastructure services to the MDM project that is, custom Publishing Service or any customized services to serve specific functions. This service may or may not have any data models defined but only workflows/processes to work upon MDM tables.

***Q 42:*** *Under what circumstances do we want it to be a document or model based service?*

**A.** **Model Based.** The model based service is a standard MDM service type which can be used to build an application.

**MDM Based.** The MDM based service is a service type which is customized for the MDM application. MDM related functions such as the automatic creation of Input staging, NetChange staging, and Output staging child-services are available in this type of service.

**Document Based.** The document based service is a service which uses the X-Documents for schema definition. You can insert documents and document views in this type of service. This helps in the backward compatibility of document-based services developed with earlier versions of MDM. You can view a list of archived service with this kind of service.

***Q 43:*** *When do you need solution setup?*

**A.** Solution setup is a substitute for schema generation and frequently used option too. With MDM 3.00.00 release, Solution Setup process within the MDM Studio is aligned with the

base MDM schema generation scripts to perform schema generation. Using this, user can upload static data as part of schema generation from within the studio which reduces development cycles not having to run batch scripts.

***Q 44:*** *What is the relationship between a UI Node Next Node Condition setting vs a Workflow Connection Property Editor Action setting?*

**A.** You need to define user actions (i.e., possible routes) in the UI Node Next Node Condition panel and can select (i.e., chanalize the routes) on the Workflow Connection Property Editor.

***Q 45:*** *What does the following files refer to?*

**A.** incr_custom_gen_schema.bat—this file includes your new services' model instances in the schema generations.

startServices.bat—this file includes your new services started with the rest of the MDM services.)

mdm-spec-gen.xml—this file includes your new services' model instances in Service/ Solution Setups.

meta-enabled-srvc-filenames.xml—this file includes your services' data cached when MDM server starts for the mentioned services.

***Q 46:*** *By default, the query search result set is limited to return 1000 records. How to configure this ?*

**A.** Using the service parameter "DEFAULT_MAX_ROWS_FETCH" in toolkit.xml, you can configure the query search result set return value. In the absence of this parameter, default value is set to 1000.

***Q 47:*** *What is headerClass, headerLink, headerStyle and headerUom? Where these attributes are used?*

**A.** **headerClass** -> Specify a style class controlling the appearance of header cell's contents.

Example: headerClass="requiredIndicatorRed"

**headerLink** -> To make header cell as hyperlink , onclick javascript operations can be performed.

Example: headerLink="true"

**headerStyle** -> Specify a style controlling the appearance of header cell's contents.

Example: headerStyle="color:yellow".

**headerUom** -> Sets the unit of measure for the field.

Example: headerUom="CAPACITY_UOM.UNIT"

More headerUom value options: DIST_UOM.KILOMETER, DL_TIME_UOM.YEARS etc.

**Usage Example**

These attributes will be supported under ui:table component.

```
<ui:table id="results" displayText="AttributeSetDetail Data"
doPaging="true" filter="true" displayTextNoRecords="No records
found" postRowIdAs="Selected_Ids"
```

```
rowId="attributeSetID#updatedLogicalColumnName" rowSrc="$root/
RESPONSE/AttributeSetDetail" selectable="All" scrollable="true">
        <ui:field-group>
          <ui:display-field displayText="attributeSetID"
id="attributeSetID" name="attributeSetID"
headerUom="CAPACITY_UOM.UNIT"/>
         <ui:display-field displayText="updatedLogicalColumnName"
id="updatedLogicalColumnName" name="updatedLogicalColumnName"
headerClass="requiredIndicatorRed"/>
          <ui:display-field displayText="description"
name="description" id="description" headerStyle="color:yellow"/>
          <ui:display-field
displayText="updatedPhysicalColumnName"
id="updatedPhysicalColumnName" name="updatedPhysicalColumnName"
headerLink="true" onClick="javascript:Test();"/>
          <ui:display-field displayText="ENTITY_STATE"
id="ENTITY_STATE" name="ENTITY_STATE"/>
          <ui:display-field displayText="SOURCE" id="SOURCE"
name="SOURCE"/>
        </ui:field-group>
        <ui:buttons id="buttons1" name="buttons1">
          <ui:button id="SUBMIT" name="SUBMIT"
displayText="Submit"/>
        </ui:buttons>
      </ui:table>
```

- AttributeSetDetail Data

| attributeSetID [UNIT] | updatedLogicalColumnName | description | updatedPhysicalColumnName | ENTITY_STATE | SOURCE |
|---|---|---|---|---|---|
| 10018 | attributeSetID11125 | | | ACTIVE | BackEnd |
| 14 | attributeSetID14 | | | ACTIVE | BackEnd |
| 8 | attributeSetID8 | | | ACTIVE | BackEnd |
| 3 | attributeSetID | | | DELETED | BackEnd |

**Q 48:** *I have heard that there is a way in Studio to tell if you are running in operational mode or analytical mode. How do you tell?*

**A.** In the Studio User Guide; Appendix H ("Publication Services"), there is a discussion on 'Publication Method'. For Analytical MDM, the drop down list will be restricted to only "Publications Database Table". For Operational MDM, all 4 choices are available.

(Also note that the Operational vs. Analytical MDM distinction has no effect on the use of Output Staging tables.)

The real distinction is more contractual license usage. That is, the Analytical MDM license is restricted to using MDM to manage data published to the Warehouse only for Analytical and BI purposes. The Operational MDM license adds to that, by expanding the use of MDM for managing data published outside the warehouse for operational purposes. We only put the technical distinction in between the 2 installers to satisfy an internal

pricing and entitlement requirement. We may elect to add other restrictions/ enhancements around this distinction in the future.

If a customer wants to use another tool to manage rules and distribution of data from the warehouse to other systems, that can be considered outside the scope of the Operational license, since MDM is not being used to manage that process or execute any rules related to that processing.

In the Release Definition, it also notes that the installers have different names and are on different CDs and ESDM files.

The Analytical MDM installer is named "mdminstall1_win_x.x.x.x.exe (BCD0-1064) and the Operational MDM installer is named "mdminstall2_win_x.x.x.x.exe (BCD0-1065).

***Q 49:*** *Is there a possibility of making the URL parameter of the Hierarchy Object editor a pick list of the table's fields?*

**A.** This parameter can actually be anything because the URL could be anything. The URL does not have to point to the table editor (but it does by default), it could point to an external web site or a workflow you have written. So the parameter corresponds to the URL you enter. Making it a pick list would limit the use of the feature.

***Q 50:*** *Can I define a hierarchy with a view as the hierarchy object?*

**A.** While it is possible to create a hierarchy using a view as the hierarchy object, operations like drag and drop through the hierarchy viewer is not possible on a view. Drag and drop internally invokes a do_db_persist request. Database operations like INSERT, UPDATE etc. cannot be supported on a database view (more so if it's a complex view).

The recommended way of defining data relationship in the case where a view acts as a hierarchy object is to define the relationship (both metadata and data) through MDRM.

***Q 51:*** *If authorization (RLS/CLS) is enabled, how does it impact when one views data through hierarchy viewer?*

**A.** Hierarchy viewer obeys RLS and data for only those rows are displayed in the viewer for which user has access. We are still evaluating the role of CLS in the viewer. As of now, viewer does not obey CLS. When a hierarchy object is created, creator can decide which column is to be displayed to the max number of 5. So in some way, column level visibility can be controlled through this feature.

***Q 52:*** *How is authorization controlled in hierarchy?*

**A.** Apart from the activity level control, in Hierarchy Viewer there are multiple levels of authorization:

a) While creating Hierarchy, it can be decided whether it can viewed for certain user group or not

b) When a hierarchy object is created, creator can decide which column is to be displayed to the max number of 5

c) We have implemented RLS in which not only node is forced not to be displayed but its children also is forced.

***Q 53:*** *Where can we get sample import template for hierarchy?*

**A.** The sample import templates are packaged at <Install_Directory>\custom\sampleApplication\testdata\HierarchyManager

***Q 54:*** *Can multiple hierarchies be uploaded through one single file?*

**A.** Multiple hierarchies can be uploaded through one single file, but if there are overlapping hierarchy objects among those hierarchies then each hierarchy has to be uploaded separately.

***Q 55:*** *Do we have a log for Hierarchy Manager?*

**A.** The hierarchy manager log (hierarchyManager.log) can be found inside the web application's domain. E.g. In the case of weblogic C:\<WL_HOME>\user_projects\domains\<Domain>

***Q 56:*** *Can the table editors tied to the Hierarchy nodes be made to conform to Favorites set up on those tables?*

**A.** Yes as long as you can construct a URL you can use that URL when you set up the hierarchy.

***Q 57:*** *How do we publish hierarchy database tables with Publishing Services?*

**A.** The process to publish hierarchy database is same as any Master Data.

***Q 58:*** *What will happen if I choose non-PK column as the linked column in linking hierarchy objects through HM?*

**A.** The viewer will not display any record in that case.

***Q 59:*** *What happens if I delete a dimension/hierarchy/hierarchy object?*

**A.** If you delete:

Dimension - the dimension is deleted along with all its hierarchies and hierarchy objects.

Hierarchy - the hierarchy is deleted along with the hierarchy object links. Please note that the hierarchy objects are not deleted if a hierarchy is deleted, only the relationship of the hierarchy object with that hierarchy is deleted.

Hierarchy object - the hierarchy object is deleted. Please note that a hierarchy object can be deleted only if it does not have any link with any of the existing hierarchies.

***Q 60:*** *What is a relational object? How are they created?*

**A.** Relational objects in MDRM are the equivalent of hierarchy objects in hierarchy. During schema generation a default relationship object is created for each master table. The primary key is considered as the linked column. If the table has composite key then all the columns participating in PK definition are considered as linked columns. If required, user can define relational objects on non-PK column by using the out-of-the box object creation UI. Please refer to *MDM Platform Server Guide* for more details.

***Q 61:*** *What is the way for me to go to the x-ref table if the x-ref table participates in hierarchy?*

**A.** The URL specified for the hierarchy object can be changed from the default hierarchy object's table editor URL to the URL of the x-ref table. Please refer to the *MDM Platform Server Guide* for details on how to edit a hierarchy object.

***Q 62:*** *What is the difference between the additional attributes that we define in x-ref and the attributes that the sample application matches to identify the golden record?*

**A.** The additional attributes that can be defined as part of x-ref creation are the properties of that relationship. E.g. Effective Start Date of the x-ref. These attributes can be used in other custom workflows that use these x-refs.

The attributes that the sample application matches on are the properties of the actual data. E.g. First Name and Last Name of a customer can be used to match duplicate customer records. These attributes are actual data of the customer; these are not properties of the x-ref between the golden customer and the duplicate customer(s).

***Q 63:*** *For Excel upload to happen is the header row compulsory?*

**A.** No , for excel upload to happen , header row is not compulsory, in that case we have to change SKIP_ROWS_IN_STD_XLS param value (from 2 to 0) in dataupload.xml , and now here, since header info is null it will pick the logical column(properties) name from template and properties sequence will be same as we define in template

***Q 64:*** *Does it read the header row and then assign the data to be uploaded?*

**A.** Yes if you are providing header row and SKIP_ROWS_IN_STD_XLS param value is not zero, then it will pick the logical name from header row(always 1st row) so if you provide wrong logical name and upload the data then it will show error on report view

***Q 65:*** *Doesn't it use the upload template for mapping excel columns to db columns?*

**A.** It will use template for mapping excel column to db column only if you don't provide header row (i.e. header row to be null)

***Q 66:*** *If someone doesn't want to put the header and just use the template how can one do that?*

**A.** For this, we can do in two ways

a) You can make SKIP_ROWS_IN_STD_XLS param value to zero and just write data in excel sheet to be upload

b) If don't want to change SKIP_ROWS_IN_STD_XLS param value , then whatever is no of skip row you need to put space in at least one column of each skip row and then put data

***Q 67:*** *Can one upload without a template?*

**A.** As per implementation we can upload but in this case header row will be necessary

***Q 68:*** *If we put more data in each row as compare to template size (table column name) then what will happen?*

**A.** Then it will skip the extra data for each row and upload successfully

***Q 69:*** *Q. What happens if you give more extra column in header row (as compared to template) and also more data for correspondence extra column of header?*

**A.** Then it will skip the extra data for each data row and header row and upload successfully.

Note: If you don't change SKIP_ROWS_IN_STD_XLS and you mention right display name in second row and your 1st row is NULL (NULL means you need to put space in at least one column of first row i.e. header row) then header information will be second row i.e. the display column name.

You can see the difference in the below XMLs:

In case you provided first header template (logical name which is hidden if you create through table editor):

```
<REQUESTS ServiceName="E2E">
<REQUEST Name="persistDBDataRequest">
<updatedLogicalTableName Value="aa"/>
<updatedPhysicalTableName Value="aa"/>
<description Value="nitin"/>
<name/ Value="nitin"/>
<SYS_ERR_SEVERITY/>
<SYS_ERR_CODE/>
<LAST_MODIFIED_DATE/>
<ENTITY_STATE Value="ACTIVE"/>
<CREATION_DATE Value=""/>
<CREATED_BY Value=""/>
<SOURCE Value=""/>
<SYS_AUTH_ID/>
<SYS_TARGET_ID Value="1"/>
<attributeSetID Value="456"/>
</REQUEST></REQUESTS>
```

In case you provide the first row as NULL and write display name in second row then second row (display name) will become header info:

```
<REQUESTS ServiceName="E2E">
<REQUEST Name="persistDBDataRequest">
<updatedLogicalTableName Value="aa"/>
<updatedPhysicalTableName Value="aa"/>
<description Value="nitin"/>
<name Value="nitin"/>
<Error Severity/>
<Error Code/>
<System Type Value="INSERT"/>
<Last Modified Date/>
<Last Modified By/>
<Entity State Value="ACTIVE"/>
<Creation Date Value="05/26/2008 15:11:08"/>
<Created By Value="USR_1"/>
<Source Value="UI"/>
<Authorization/>
<Target Id Value="11"/>
<attributeSetID Value="R1"/>
</REQUEST></REQUESTS>
```

***Q 70:*** *For the above case, if display names are referred as header info, is the request getting executed successfully as display names may have blank spaces in their name?*

**A.** Only that display name value is saved for which display name is same as logical name. It considers the display name as logical name, so that column value is saved where display name is same as logical name.

***Q 71:*** *Excel uploads work only for Master tables. Can we have the option to load into Input Staging so that the data will go through the Business Rules? Any plans for that type of functionality?*

**A.** In MDM 2.01 Business Rules have been enhanced so that excel upload goes through Validation, Update and Insert rules while uploading to master. Right now there is no plan of supporting excel upload on input tables.

***Q 72:*** *What is the lifecycle of the Excel files that are uploaded?*

**A.** Once uploaded successfully, excel files remain in archive folder unless deleted explicitly.

***Q 73:*** *Are they going to be deleted by the application once uploaded to the database?*

**A.** No. After successful upload, excel file moves to the archive folder from the inprocess folder.

***Q 74:*** *What does each of the folders - archive, inprocess and source do?*

**A.** source: excel that needs to be uploaded can be stored here. If the folder has a marker file named as autoload.txt then the upload process initiates automatically at the booting time of MDM server.

inprocess: Upload initiation process puts the excel to be uploaded under inprocess directory and when upload completes, it moves excel to archive folder (deletes from here). This would only happen when upload process completes successfully. If there is any interruption (say abrupt failure of MDM/AppServer) then the file remains in inprocess folder and next time when MDM server boots up, it tries to upload the pending jobs from this inprocess folder.

archive: successful upload results in moving excel from inprocess to archive.

***Q 75:*** *What will happen if a file is being uploaded to the database (i.e. in the inprocess folder) when the app servers stop? Will the application recover upon a restart?*

**A.** Yes. It would try to upload the excel when MDM server restarts.

***Q 76:*** *Is there a way to stop/abort the upload process through the application UI?*

**A.** No.

Business Rules/Do_Db_Persist

***Q 77:*** *Is it possible to generate a .pdf document of the Business Rules/SQL Rules/DoPersist Validations? Like we do with the data model, and like Oracle Data Integrator (formerly Synopsis) does?*

**A.** Not currently. All the business rules defined through the new Business Rules UI are stored in a table DTA_RULE_TMP. A simple select query on this table will return all the rules.

***Q 78:*** *Is it possible to generate a .pdf document of the Workflow/Rule logic? Again like we do with the data model and Oracle Data Integrator does?*

**A.** Not currently.

***Q 79:*** *Once a record is soft deleted how can it be reactivated? Currently after deleting something they try to recreate it, and of course with a soft delete they cannot do that, so need a way to reactivate.*

**A.** In MDM2.1 we are introducing ways to reactivate and perform hard deletes. Before insert do_db_persist tries to find if the record for the given unique id already exists and if yes, the SYS_ENT_STATE state of the record is changed from DELETED to ACTIVE.

***Q 80:*** *In the Table Editor Activity Control part of Security Management is it possible to create custom groups of tables to ease the administrative management of Table Editor Authorizations? (Like what is there for all IN, NC, MST, OUT tables - but custom for tables a,b,h,q,r,s...)*

**A.** The left navigation is dynamically constructed for table editors based on the way the model folders are created in studio. So one can create them accordingly during design phase.

***Q 81:*** *I have heard that Table Editors can be made to work with views. The question now is: do Table Editors work with views for viewing data only or also for updating data?*

**A.** Table editors can work on any db entity that has an xdoc available in MST service. The underlying entity can be a table or view, the framework does not bother. Inserts and Updates would work as usual. So if the db allows it, it will work fine. E.g. DB will not allow DMLs on a complex view, so that will not work.

***Q 82:*** *What is the use of the "Where Used" icon in the Table Editors?*

**A.** Let's say table A is being referenced from Table B and C. Now if you select a record in Table A and click on where used, you can get a report as to which records in Table B or C are using this record in Table A.

***Q 83:*** *Can derived fields be added to the Table Editor listings? (Like today's date - Birth date * 365 to create a new "Age" column for example.)*

**A.** No table editors only support fields defined in the model.

***Q 84:*** *Can you use security to disable specific buttons in the Table Editor (specifically Create) or does it just affect the Table Editor completely? (Looks like the answer is yes by using the Table Editor Activity Search tab under Manage Roles, but there is not a 1-1 relationship between the TE buttons and the Create/Edit/Delete choices so we want to double check.)*

**A.** Yes. Roles need to be defined explicitly.

***Q 85:*** *If I have mentioned data base details (database name, host name, user name, password etc.) in any of my custom workflow files, will Deployment Manager automatically update such details while deploying?*

**A.** No, Deployment Manager only updates the DB details in the custom service files. For any other custom workflows, user has to manually change the details. It is not recommended for user to use DB details directly inside workflows, they should extract those details from the xserver.

***Q 86:*** *What changes need to be made to the incremental schema generation files when creating new ones for new services? (Comparing the incr_gendb.bat file with the incr_custom_gendb.bat file I see only the called file names the relative directory paths have changed. Anything else? Anywhere this is documented?*

**A.** In most implementations one would add a customized service to toolkit (child service to toolkit - similar to Sample Application) and not add a brand new service! Usually implementations would add new models, workflows and rules rather than complete new services. If only this is done then no changes are required.

***Q 87:*** *When you add a column to an already populated table and then do an incremental schema generation is the data in that table wiped out or is a new column simply added to all existing records?*

**A.** The new column is only added to the existing table.

***Q 88:*** *What is the best way to sequence the various MDM upgrades (Dev, then Cons/Staging, then Prod), are there any best-practices around the use of the new Deployment Manager to get the environment set up on each subsequent system?*

**A.** Deployment Manager helps in uploading and deploying the custom application across staging and production environments.

Typically in the dev environment, users would use Studio and upload the custom application to the deployment database.

User would then run the Deployment Manager installer and deploy the custom application to the staging environment. Once testing is complete and satisfactory, user can perform the same steps on Production. For more details on Deployment Manager, please refer to *MDM Platform Studio User Guide*.

***Q 89:*** *In collapsed tier, how is debugging managed in development mode?*

**A.** In development mode it has to be co-located with all the components including locator.

***Q 90:*** *In terms of role assignment, Is there any difference in behavior between RLS/CLS?*

**A.** In RLS, no role assigned means "access to all" while in CLS it's "access to none".

Another difference in behavior between RLS and CLS is while in CLS we can assign multiple roles to a column, the same is not possible in the case of RLS.

***Q 91:*** *What happens if CLS is enabled and user does not assign roles to PK and Index columns?*

**A.** Right now, we are keeping the PK auto-selected when the assign page is loaded so that PKs always get assigned. We cannot un-assign a role from a PK. If the table has index defined on any column, that column must also be selected while assigning CLS. Unlike PK, index columns are not auto-selected, so user must manually select them while assigning roles to the columns.

We have CLS enabled on a document. A workflow needs to access a column from this document that the user does not have access. The workflow needs to get the value of that column and update the same into another document. What happens if CLS is enabled in this scenario?

This is not possible now. In this scenario the workflow will fail throwing an exception.

***Q 92:*** *This requirement is similar to that mentioned in the previous point except that in this case even if user knows the column he cannot assign the role to the column because business case demands that. For example, say I want to implement a pricing calculation solution. Typically, what I would expose to the user on UI is the final calculated price; the columns that I require to arrive at the final price will not be shown to the user. How can this be achieved through CLS?*

**A.** The best way to implement such requirements is typically to build a PGL workflow to show only the final calculation and use the other columns behind the scene. However, if user wants to build a UI where Pricing Department wants to view all columns but the end users will only see the final price and control the visibility through the use of CLS, currently this is not supported.

***Q 93:*** *I created two linked tables (Parent & Child) with a FK link. The FK linked column of child table (which is a non PK column) has source as MASTER. When you run e2e giving a null value in the FK linked column, e2e fails as we don't have that particular column in NC and being a FK it expects the column in NC. What am I doing wrong?*

**A.** This is the expected behavior. Any table participating in E2E should have dependant tables as part of the load definition.

***Q 94:*** *I defined a column, which is not PK, as 'NOT NULL'. After solution set up the column in the corresponding input staging table is not showing as 'NOT NULL' column for table with SOURCE="BOTH". Is this wrong?*

**A.** This is as per design. If the table class is BOTH and field facets are ignored, only PKs would be participating in NC. All constraints on the input staging tables are disabled to allow unconstrained data to be populated. Hence the not null constraint is also disabled. In this particular case if this field is not sourced from Backend, it should not have the NOT NULL constraint on it.

***Q 95:*** *Is there any way to change the Audit Trail functionality so that only specified changes (such as to columns a-h but not to columns i-k) would be captured?*

**A.** The current audit trail capability is at a document level and not at individual column level. So currently there is no way to do this.

***Q 96:*** *For Code Master table can we have Excel upload capabilities like we do for the Master tables?*

**A.** Currently there are no plans to support this.

***Q 97:*** *Can we have effective date range functionality for "Code Master Values"?*

**A.** Currently this is not possible.

***Q 98:*** *We have been trying to get MDM up and running against an IBM WebSphere 6.1 Server on XP. We have followed the Installation instructions for Web Client Deployment and have everything up. But we don't know what port number to enter to get to MDM, // localhost:7001/mdm doesn't work. Any idea?*

**A.** The default application port for WebSphere is 9080. However, this port number can vary depending on the number of app servers that is created. Also the port number can be edited by using the WebSphere admin console. For details, please refer to the orange book on deployment of MDM on WebSphere.

***Q 99:*** *Can E2E be programmed to run automatically or does it have to be run manually from the UI?*

**A.** E2E can run from a batch script (see the scripts in C:\Teradata\MDM\3.0.0\batch), which can be scheduled via a scheduler.

***Q 100:*** *We have few tables where the ID is surrogate (generated sequence number). When we create the record from table editor on these, the expectation is that the ID will come back with next number in the queue and should be non editable. Currently it does not and asks user for the value. Is this designed behavior? Is there any ways to show generated ID on UI?*

**A.** The behavior pre MDM 2.0 for MST table is as follows -

1. When user clicks on create, PK (if it is a surrogate id) is not shown on UI.

2. User enters other values and clicks on save.

3. Framework invokes DO_PERSIST API. If no specific save API is defined then ADD_DOC is invoked. ADD_DOC generates an id in this case and uses that for save.

The behavior in MDM 2.0 and 2.0.1 for MST table is as follows -

1. When user clicks on create, PK (if it is a surrogate id) is not shown on UI.

2. User enters other values and clicks on save.

3. Framework invokes DO_DB_PERSIST. Here it does not invoke ADD_DOC but performs DB friendly inserts directly! So no ID is generated despite the fact that it has been defined as a surrogate key and the insert fails. This could be an issue if someone is using TE!

In MDM 2.1 we have changed this.

Step 1 and 2 are same.

In Step 3, Framework generates a unique id (if it is a surrogate key) and then invokes DO_DB_PERSIST.

***Q 101:*** *How long does it take to snapshot data from Master to Output Staging? Of course it depends on the size of the tables but is there a rule of thumb such as X seconds per x thousand records?*

**A.** MDM executes a procedure that internally executes a query like this - insert into _ _ _ select * from _ _ _.

We can publish the standard database performance numbers of such a query.

***Q 102:*** *I have run into an error when running MDM on a laptop that has TD 12.0 installed. The error is "insufficient memory to parse this request" when running the Data Load workflow in the sample application. This shows up as Net Change stuck in the "Started" status (not Completed). When you look in the E2E.log you will see this error from Teradata.*

**A.** The steps to fix this are:

1) Open the Teradata Database Window (Start->Programs->Teradata Database->Database Window).

2) Click on the Supvr icon to open the Supervisor window.

3) Enter "dbscontrol" at the prompt in the Supervisor window to open the dbscontrol window.

4) Enter "display performance" at the prompt in the dbscontrol window.

5) Find "MaxParseTreeSegs" in the list and note the number (probably #4).

6) Enter "modify performance 4=2000" at the prompt in the dbscontrol window (where 4 is the number for MaxParseTreeSegs).

7) Enter "write" at the prompt in the dbscontrol window.

8) Enter "display performance" at the prompt in the dbscontrol window and make sure your new setting is there.

9) Enter "quit" at the prompt in the dbscontrol window, the supervisor window and exit the Database Window.

10) Shut down and restart MDM server, WebLogic Server, and Teradata.

***Q 103:*** *Do you have any example command (to issue from a Windows/UNIX command line) that you could share with us about running MDM functionality (E2E workflow, other workflows) from a scheduler?*

**A.** To schedule a new task

1.Open Windows Start--> Settings --> Control Panel --> Scheduled Tasks.

2.Double-click Add Scheduled Task.

3.Follow the instructions in the Scheduled Task Wizard.

Notes

"To open Scheduled Tasks, click Start, point to Programs, point to Accessories, point to System Tools, and then click Scheduled Tasks.

"If you want to configure advanced settings for the task, select the Open advanced properties for this task when I click Finish check box on the final page of the wizard.

"Confirm that the system date and time on your computer are accurate, because Scheduled Tasks relies on this information to run scheduled tasks. To verify or change this information, double-click the time indicator on the taskbar.

"If you leave the password blank and you want the task to run when you are logged on, open the task. On the Task tab, select the Run only if logged on check box. The task will run at its scheduled time when the user who created the task is logged on to the computer.

In the Windows scheduler user can select any of the bat files provided with MDM software - e.g. <INSTALL_ROOT>\batch\startDataLoadWorkflow.bat - to schedule the E2E workflow.

***Q 104:*** *I have non-Latin characters in my database (character set for the column is defined as UNICODE). When I view the data, they are shown as "junk" characters. What should I do so that the special characters are displayed properly?*

**A.** For special characters to show up properly on MDM UI, the following changes need to be made:

1.Open the xserver.xml (<Install_Directory>/cfg/properties) and change the JDBC string to UTF8 (You can also try UTF16 if UTF8 doesn't work). It should look like:

<db-url Value="jdbc:teradata://153.65.183.86/TMODE=TERA,CHARSET=UTF8"/>

2.Make the same changes in xserverweb.xml located in the same directory.

3.If you are using WebSphere, refer to the *"Deployment on WebSphere" section in MDM Installation Guide.*

***Q 105:*** *Can we import relationships from Erwin? or Other applications?*

**A.** You can import a data model from Erwin, a relational database or from XML. This will contain foreign key relationships, but not really hierarchy relationships. Generally, hierarchy (data) relationships are contained within the data... So the model will contain a third normal form data relationship between two (or more) entities using a relationship table, but the "active" relationship is the data within that relationship table. So, that model (tables) can be imported from Erwin (etc.) and the data can be imported using E2E, Excel, etc.

***Q 106:*** *Why do we need both the Structured and the Unbalanced types of hierarchies?*

**A.** Both of these hierarchies do contain level information (nodes belong at certain levels and cannot change levels). But Ragged is the more generic of the two, so it can support both. Unbalanced is a pure parent-child relationship and does not use the concept of levels, so it is fundamentally different.

***Q 107:*** *Why do we get this error during E2E*

<td>Error while executing DO_DB_PERSIST command

<RESPONSE Status="Error" Description="[NCR] [Teradata DBMS] [TeraJDBC 12.00.00.01] EXECUTE_DOPERSIST_DB_DATA_MMT:Column 'CUST_ID' is NOT NULL. Give the column a value.

[BCM_MASTER]"/>

**A.** When the Source is set to "Both" at table level, you need to explicitly set the source of Property in Property Facet tab to "BackEnd".

***Q 108:*** *In Hierarchy Manager cut and paste when you go to paste what if you change your mind and don't want to paste? (The only option in the dialog box is OK.) Is there a way to abort, or undo, or would you have to just reverse cut and paste?*

**A.** The Search Dialog in the Hierarchy Viewer allows you to click on Cut and then select a node. This node becomes marked for cut/paste. If you change your mind, there is a button in the dialog labeled "Clear". Just click that and the node is removed from the "Marked node:" area (both in the dialog and in the Viewer. Now, once it has been cut and pasted, there is no undo.

***Q 109:*** *When building a business rule, in Expression Builder, Regular Expression, the bottom property reference is always the first property alphabetically and does not change even if you change the property in the top part. Is this a bug?*

**A.** No, this is not a bug... The bottom property reference becomes available to change if you select the "Field" radio button. This means you want to compare the top property, using the selected Condition, to a field in another or the same table (rather than to a specific Value).

***Q 110:*** *If a stored procedure is added to the library of existing stored procedures will it be picked up by the drop down list of the Admin UI/Stored Procedures section?*

**A.** The Stored Procedures are picked up from the available compiled SP's in the MDM USER's database. Here is the query used to get the list for the drop down box:

SELECT tablename FROM dbc.tables WHERE databasename = mdm_user and tablekind = 'P' ORDER BY tableName ASC;

You can then select one of those SP's to add to the list of available SP's to execute using the Add button. When they are added to the Execute list, they are simply placed into the MDM SYS_STORED_PROCEDURES table.

***Q 111:*** *When using OMI, if there is a deletion in the Erwin/RDB/X-doc specification will the object then be deleted in the MDM data model too? Or is OMI just for model adds and changes?*

**A.** OMI is just for model additions and changes

***Q 112:*** *Order of execution of Business Rules: Filter, Insert, Update, Validate or Filter, Update, Insert, Validate?*

**A.** If there are Filter, Validate, Insert and Update rules for a table, then the order of execution will be Filter, then Validate, then Insert or Update.

***Q 113:*** *How to best backup/export-import business rules that were created via the Business Rules editor?*

**A.** To export Business Rules, you would need to execute/run Export_SQL_Validation_Rules.bat/sh file under <MDM_Install>\bin folder. This will export Business Rules in a file named sql_validation_rules.xml under <MDM_Install>\cfg\xservice\toolkit\data folder.

Note: In case of custom application, execute custom_export_SQL_Validation_Rules.bat/sh from <MDM_Install>custom\appName\bin

You will have to post this file onto MDM server to upload them again.

Refer to *Incremental Schema Generation section of MDM Platform Server Guide* for more details.

***Q 114:*** *How to have multiple activities for a button tag.*

**A.** Points to remember while using the activity attribute with button tag:

- Activity should be single pipe symbol seperated rather than double pipe. For example: activity="WFDS_PACE_NPP_ADMIN_ACTIVITY||WFDS_PACE_EBR_RW_ACTIVITY" should be like activity="WFDS_PACE_NPP_ADMIN_ACTIVITY|WFDS_PACE_EBR_RW_ACTIVITY"
- In button tag, activity attribute should contain activityIDs. If activityName is provided rather than activityID then it will not match, even if active role has that activity.
- Only if the button tag have activity attribute then a check will be performed in following manner:

Get the active Roles --> find all activites listed for them, say List A. Perform check if any of activity, listed in button tag's activity attribute, matches to List A, then the button will be visible otherwise not.

***Q 115:*** *Business Rules not getting displayed properly.*

**A.** Perform the following checks:

- Enable the browser to use Java 1.5.0_13 for applets. On Internet Explorer, click on Tools > Internet Options > Advanced tab > select the “Use JRE 1.5.0_13 for <applet>” option.
- Remove the folders created at a location similar to the one below:

  *C:\Documents and Settings\<username>\Application Data\Sun\Java\Deployment\cache\6.0*

  Please make sure none of the java applications are running when you do this and delete all the folders numbered 0 onwards and folders with non-numerical names be retained.

***Q 116:*** *How to set the value of an XML variable as a Line Separator (New line)?*

**A.** Use system-property as below. It returns Line Separator value as '\r\n'

```
<SET Var="lineSeparator" FromSelect="system-
property('line.separator')" />
```

***Q 117:*** *How to clear all the form fields on the PGL page?*

**A.** Create a new button with its onclick function as clearAllFields as shown below:

```
"<ui:button id="Clear" name="Clear" displayText="Clear"
onClick="javascript:clearAllFields()" />"
```

On click of this button, all the form fields will be cleared.

***Q 118:*** *How to revert to original values on the form fields on the PGL page when the page is loaded?*

**A.** Create a new button with its onclick function as resetAllFields as shown below:

```
<ui:button id="Reset" name="Reset" displayText="Reset"
onClick="javascript:resetAllFields()" />
```

On click of this button, all the form fields will be reset to its default value.

***Q 119:*** *How to increase the socketpoller timeout at ISG?*

**A.** Specify the Timeout parameter in SocketPoller command in ISG scripts (incr_static_data_upload.bat/sh). For example:

SocketPoller Command in the OOTB script ---> %JAVA_HOME%\bin\java com.teradata.xcore.util.SocketPoller localhost xserver.xml

Change to --> %JAVA_HOME%\bin\java com.teradata.xcore.util.SocketPoller localhost xserver.xml 20

Third parameter is the timeout period in minutes.

***Q 120:*** *Why MDM ID generation uses separate database connection and not from connection pool ?*

**A.** By Design, the BlockTableIdGenerator needs at least two database connections in the connection pool to avoid deadlock.

It assumes Xcore has two pools to give out connections. One Pool for regular transactions and seperate pool for IDGen purpose. (Default Pool Size 1).

XCore has different pool for IDGen connection and it gives the connection from that pool and BlockTableIdGenerator has to release the connection after it's usage.

***Q 121:*** *The Business rules page does not show custom tables in the rule creation page without the Tomcat class path settings for customcoloc.jar*

**A.** For collapsed mode, the following jars have to be explicitly added to the application server (Websphere/Weblogic/Tomcat) classpath in the same order.

1. *<MDM_Install_Diectory>/web/mdmclient/WEB-INF/lib/customcoloc.jar* (in case of custom/sample application, this has to be added first).

2. *<MDM_Install_Diectory>/web/mdmclient/WEB-INF/lib/coloc.jar*

For details on manual collapsed mode setup, refer to *Appendix Collapsed Tier Setup in MDM Platform Studio User Guide.*

***Q 122:*** *How to set EXPORT_ALL for a table in the workflow.*

**A.** Perform the following steps set EXPORT_ALL for a table in the workflow.

1. Create a hidden parameter in the workflow(.pgl) as following with "EXPORT_TABLE_ALL_RECORDS" id.

<ui: hidden-field id="EXPORT_TABLE_ALL_RECORDS" value="yes"/>

2. Set export parameter true in the table.

3. Set the "start count" and "max rows" for the request/GET_DOCUMENT as following:

```
<SET Var="maxRow" FromValue="20" />
<IF_TEST Test="$thisParam/EXPORT_ALL/@Value='yes'">
<THEN>
<SET Var="paginateStart" FromValue="0" />
<SET Var="maxRow" FromValue="{$thisParam/MAX_ROWS/@Value}" />
</THEN>
</IF_TEST>
<GET_DOCUMENT Name="DATA_RULE_TEMPLATE" ServiceName="BPE_META"
AssignToVar="results" StartAtRow="{$paginateStart}"
MaxRows="{$maxRow}" ReturnRowCount="yes" DirtyRead="true"/>
```

For more details, refer OOTB workflows (Configurable UI).

# Index


**A**

Activate
- User group 550

Add
- Code master type 558
- Code master value 556
- Contact 551
- Contact Info 552
- Organization 537, 570, 571, 572
- Role 540
- User group 546

API_DOC IDL 507
Arithmetic Functions 396
Assisted Upload 485
Attributes
- about-action 199
- action 40
- activity 32, 35, 44
- align 44, 53, 159
- anyFieldHasErrors 41, 99
- anyFieldIsRequired 41, 99
- app-url 199
- axis 181, 182
- background 110, 199
- callbackData 67
- cellName 184
- cellSpacing 152
- changePwd 110
- checkBoxEnabled 190
- checkedValue 53
- class 44, 53
- collapsable 32, 41, 144, 190
- collapseImage 190
- color 67, 176
- colSpan 44, 53
- column 146
- columns 44, 104
- command 110
- configurable 44, 70
- containerId 107, 144
- content-url 111, 199
- count 113
- dataType 44, 53, 159
- decimals 44, 53, 163
- defaultValue 44, 54, 76
- depth 146, 149
- disabled 44, 49
- displayCondition 32, 35, 44
- displayNoWrap 44
- displayPage 46, 49, 96
- displayText 32, 35, 41, 44, 425, 427, 429, 430
- displayTextNoRecords 144
- document 144
- doPaging 144
- draggable 145
- editable 32, 35, 44, 54, 184, 425, 427, 429, 430
- elementName 46, 54
- emphasized 49
- enableBySelections 49
- error 111
- errorDescription 111
- expandImage 191
- exportable 179
- filter 54, 185
- forced-logout 199
- format 44, 54, 89, 185
- formula 184
- framed 199
- freeze 144
- from 111
- fromDate 100
- headerClass 44, 54
- headerDataType 44, 54, 185
- headerEditable 44, 54, 64, 185
- headerElementName 46, 54, 72
- headerLink 44, 54, 59
- headerStyle 44, 54
- headerType 45, 54
- headerUom 45, 54, 59, 185
- height 32, 107, 175

help 41, 144
help-action 199
hideFields 81
id 32, 35, 41, 43
imageActive 175
imageInactive 175
inFormat 45, 54, 59, 185
inner 32
instructionMessage 99
invert 144
leafImage 191
linked 144
logo 111, 199
logout-action 199
massEntry 45, 54, 59, 71
maxlength 70
maxRows 144, 179
message 102, 108
method 41, 144
name 41, 43
nav-url 199
noChildren 146
numericPattern 45, 54
onChange 70, 88
onClick 35, 45, 49, 425, 427, 429, 430
onlyInHeader 76
onlyJavascript 76
operator 103
optionSrc 88
Page 131
panelImage 111
postAction 191
postRowIdAs 144
protect 183
rememberSelections 144
repeat 184
required 45, 55
rowId 144
rows 45, 81
rowSpan 45, 55, 147
rowSrc 144, 175
rowSubTreeSrc 144
rowTreeSrc 144
scrollable 32, 35
select 35, 45, 183, 425, 429, 430, 432
selectable 180
selectAll 88
selectColumnText 180
selected 51
selectedStep 32, 35
selectedStyle 175
selectOne 88
severity 103, 108
sheetName 183
showDivider 81
showRefreshIcon 104
size 45, 55
sortable 45, 55, 90
sortBy 145
sortOrder 145
source 104
src 65, 120
startAtRow 145, 180
Style 133
style 45, 55
submitAsXML 49
submitTo 49
subRowSrc 172, 180
successMessage 99
target 41, 49
targetName 125
targetType 65, 125
test 162
title 104
toDate 100
totalRowCount 145, 180
type 32, 45, 55, 185
unFormat 46, 55
uom 46, 55
uploadType 46, 55, 60
url 118
urlParam 145
user-name 111, 199
validationFunction 108
valign 154
value 46
valuePrepend 46, 55
width 32, 107, 154, 175
withField 103
withValue 103
xLablesWidth 107
xslfile 104

yAxisNumTicks 107
Audit Trail 218
**B**
BlockIdGeneration 489
blockSize 490
Body 467
Boolean Functions 393
**C**
CALL_EVENT 497
CALLBACK_ACTIONS 462
CALLBACK_DATE 462
CALLBACK_DURATION 462
callBackDateVar 462
callBackDurationVar 462
child element 3
Child Tag 338, 509
Child Tags 285
Comparison Functions 398
Container 30, 32
Tabbed container 30, 33
Create
Auth views 568
Cross Reference 533
Filters 529
Global temp tables 568
Hierarchy 535
Relational object API 526
Relationship 531
User group 547
**D**
Date and Time Patterns 411
Delete
Business rule 567
Contact 554
Cross Reference 534
Filters 530
Hierarchy 536
Relational Object API 528
Relationship 533
delimiter 490
Display Field 10
document 489
Document Structure 4
Documentation vii
documentcallBackDateVar 462
document-wide 4
DocView 692, 696, 701, 710, 713
doPaging 136
durational timer 461
**E**
Edit
Relational Object API 527
Element Attributes 4
Element Containment 3
Element Realization 4
E-mail service 664
EMx 494, 495, 497, 500
ERROR 370
Error 474
Event 491
Defining 491
Subscribing 493
event 491, 493, 495, 498, 500
Event Descriptors 641
EVENT_DESCRIPTOR 491
EVENT_DOC 491
Export Framework 577
Key features 578
Expression 258, 474
external documentation 3
**F**
Field Group 10
Filterable 36
FilterExp 474
FilterExp/FilterName 474
FilterName 474
FOR_DOCUMENT 462, 638
Framework Service Operations 369
DO_DB_PERSIST 371
UPDATE_VALIDATION_RESULT 370
VALIDATE_DOC 369
Functions
Arithmetic functions 396
Boolean functions 393
comparison functions 398
Java Functions 418
Miscellaneous Functions 414
Node-Set functions 386
Number function 394
String functions 388
time related functions 402

**G**
Get
Assignable roles 540
Code master description 557
code master value 557
Contact 553
Countries API 570
Document names 569
Organization 538
Property names 569
Publication objects 567
Roles for user 541
Service names 569
User 545
User group 548
User group details 549
User groups 548
User role 542
Users 544
Global Variables
define 720
GUARANTEE_LEVEL 467
Guaranteed Messaging 467
Guaranteed Request invocation 467
guaranteed timer request 461
GUARANTEED_REQUEST 467
**H**
handler 490
Header 467
help document 3
**I**
identifiedByDoc 462
image resource 3
Input Parameters
Cross Reference API 534
Enterprise Model API 554
Filters API 530
Manage Objects API 528
Relationship API 533
Xcore API 566
Integrated development environment (IDE) 202
INVOKE_VALIDATION_FRAMEWORKrule 480
IsActive 474
IsFilter 474
IsNoCache 474
**L**
Link Data 532
Logical Operations 326
EXIT 334
FOR_EACH, BREAK, CONTINUE 330
IF_TEST, THEN, ELSE 326
IF, THEN, ELSE 328
REPEAT 331
WHILE 332
**M**
Manage Relationship APIs 526
Master Data Manager
Elements 202
Studio 202
MDM Elements 203
X-Document 205, 207
Document Constraints 218
Document Identifiers 208
Document Index 210
Document Links 214
Document Properties 211
Foreign Key Constraints 216
Overlay Documents 217
Surrogate Key 210
X-Rules 205
Business Requests 227
Condition Statements 225
Special Variables 224
Variables and Expressions 223
X-Server 203
X-Service 204
X-Workflow 204
MDM framework 203
Message Events 654
Message Template 655
Modify
Code master type 558, 559, 560, 561, 562, 563, 564, 565
Code master value 556
Contact 553
Organization 538
User 543, 544
User group 547
MSG_KEY 467
multiple table updates 692

**N**
Name 473
name 489
nextDoc 473
nextDocList 473
Node-Set Functions 386
NUM_RETRIES 467
Number Functions 394
**O**
Object Association 3
  Association by Element Containment 3
  Association by URI 3
  Association by URL 3
On Failure actions 467
On Success actions 467
ON_FAILURE 467
ON_SUCCESS 467
**P**
Page layout 2, 3, 4, 9, 30
  Button 46
  Button Group 49
  Cell 153
  Chart 105
  Chart Field 66
  Check Box 51
  Choose Field 156
  Compare 102
  Container 10, 30
    element 30
    Simple Container 30
    Tabbed container 30
  Custom Validator 108
  Date 99
  Display Field 61
  DIV 72, 73, 200
  Divider 83
  Drop Down 84
  Dynamic PGL 167
  Entry Field 67
  Excel Pivot 169
  Field Group 77
  Field Space 81
  For Each 109
  Generic Notes 8
  Grid 151
  Header 94
  Help 95
  Hidden Field 75
  Horizontal Rule 92
  Horizontal Tree 172
  Image 119
  Image Link Field 121
  Legend Field 176
  Link 124
  Link Field 126
  Links 130
  Login 109
  Navigation 111
  Non-Breaking Space 113
  Option 90
  Otherwise 164
  Pad 114
  Pad Item 116
  Page Group 103
  Pivot 177
  Pivot Data 180
  Pivot Field 181
  Radio Button 56
  Root 107
  Row 152
  sample projects 446
  Script 155
  Search 36
  Shell 199
  Sort 118
  Spreadsheet 182
  Spreadsheet Field 183
  Spreadsheet Row 186
  Step 10, 33
  Table 133
  Text Area 41
  Tree Node 187
  Treecell 145, 148
  UI DIV 132
  URL 130
  Validation 96
  Vertical Rule 93
  Vertical Tree 188
  When 161
  Wizard 30
  Workbook 191
Page-able 36

Page-Group 4, 10
Pagination 136
parent element 3
prefix 489
previousDoc 472
previousDocList 473
Purge Framework 590
  Key Features 591
Purpose iii
**R**
RAISE_EVENT 492, 497
reference-able 3
RETRY_INTERVAL 467
Running Rules 505
**S**
Save-able 36
Scalar Functions 280
Search Framework 599
sequenceName 489
Server
  Run time 203
Service Name 474
Services Based Architecture 202
SEVERE_ERROR 370
Severity 474
simple timer callback request 461
Smart XCommands 274
  ADD_DOC_SMART 274
  GET_DOC_SMART 274
  MODIFY_DOC_SMART 275
  REMOVE_DOC_LOGICAL 275
Sortable 36
SQL Operations 276
  EXEC_PROC 284, 286
  EXECUTE_BATCH_SQL_DML 283
  EXECUTE_SQL_DDL 284
  EXECUTE_SQL_DML 282
  EXECUTE_SQL_QUERY 281
  SQL_FN 276
ST_Geometry 213
START_DATE 467
State Model 622
State transition 689, 690
Step 33, 34
step 33
String Functions 388
SUCCESS 370
system-to-human 2
system-to-system 2
**T**
TableIdGeneration 488
template 485
Teradata UI guidelines 3
Test service setup 668
thisContext 231
thisDoc 231, 462, 472
thisField 472
thisFieldDoc 473
thisParam 231
thisParent 473
thisReturn 231
Time Functions 402
TimeIdGeneration 488
Timer 460
  Commands 462
  Examples 464
  Functions 464
  Poll Interval 461
  Setup and Callback 461
Troubleshooting 446, 457, 472
**U**
UI Element
  Container 3
  Search 3
  Step 3
  Table 3
UI Related Operations 335
  Child Tag Attributes 338
  CREATE URL 350
  DROP_DOWN_OPTIONS 349
  FORM_GET 340
  FORM_REMOVE 342
  FORM_SEARCH_FILTER 337
  GET_SESSION 337
  PAD 335
  PAD_ITEM 335
  SET_SESSION 336
  TAG_ERRORS 346
UI Services 2
Unassisted Upload 486
Update
  Business rule 568

Filters 529
URI association 3
User defined variable 473
Utility Operations 351
DO_TRANSACTION 359
EVAL_EXPR 369
EXCEPTION 357
EXEC_INFORMATICA 360
EXECUTE_SHELL_COMMAND 353
EXECUTE_TRILLIUM 362
GET_CURRENT_TIME 356
HANDLE_SYSTEM_EXCEPTION 356
INSTANCE_FUNCTION 367
INSTANCE_METHOD 368
LOG 353
NATIVE_FUNCTION 366
NOTIFY 368
PRINT_TO_FILE 355
PRINT/PRINTLN 352
REFRESH_FROM_CACHE 359
SET_LOGICAL_DATE 356
START_WORKFLOW 351
STORE_XML_FILE 355
**V**
Validate
Activity for users 546
Login 550
Role 542
Role for user 542
Validation Framework
Attributes 473
Variables 472
Validation framework 472
VFS validation 524
View hierarchy 36
view-hierarchies 3
**W**
Workflow Nodes 642
**X**
x_rule 473
Xcore 522, 573
X-Document
Referencing Declarations 512
X-Document Operations 247
ADD_DOCUMENT 247
CREATE_DOCUMENT_ID 249
Expression 258
FETCH_DOCUMENT_LINK 268
GET_DOCUMENT 254
GET_DOCUMENT_LINK 270
GROUP_BY 261
MODIFY_DOCUMENT 252
NOT_EXIST_QDL 264
ORDER_BY 259
PROCESS_GET_DOCUMENT 272
QUERY CRITERIA 250
QUERY_DOCUMENT_LINK 262
REMOVE_DOCUMENT 251
SAVE 253
SELECT + QUERY_DOCUMENT_LINK 266
X-Documents 203
features 205
XML Manipulation Operations 287
ADD_CHILDREN 299
APPEND_CHILDREN 302
CLONE_XML 319
COLLATE_XML 309
DELETE_ALL_CHILDREN 308
GET_TEXT 298
GROUP_DOCS 314
MAP 312
MERGE_DOC 316
MERGE_ERRORS_DOC 318
REMOVE_CHILDREN 303
REMOVE_NODES 304
SET 287
SET_CHILD 298
SET_PROPS 288
SORT 324
SORT_DOCS 320
STRING_TO_XML 318
TO_DOCVAR 289
XML_TO_STRING 319
xmlns 4
X-Path 379, 423
Basics 379, 423
Expressions 383, 430
X-Rules
Business Requests
DEFINE_INIT 228
DEFINE_LISTENER 228

DEFINE_METHOD 227
DEFINE_NOTIFICATION 227
DEFINE_POST_CREATE 227
DEFINE_POST_MODIFY 227
DEFINE_PRE_CREATE 227
DEFINE_PRE_MODIFY 227
Condition Statements
Compound Condition 226
Simple Condition 225
features 205
Invoke Rules 243
REQUEST 243
Invoke RulesSEND_NOTIFICATION 244
Special Variables
Global Variables 225
REDIRECT_URL 224
Service Variables 224
userName 224
XSD validation 524
X-Server 203
X-Service 204
features 204
XSL Transformations 3
X-Workflow
features 204