

Radware vDirect

CONFIGURATION TEMPLATES AND WORKFLOWS QUICK START

Document ID: RDWR-vDIR_TPWF_QS2005 Rev 13

Software Version 4.12.0
May, 2020

TABLE OF CONTENTS

Important Notices	7
Copyright Notices	8
Standard Warranty	12
Limitations on Warranty and Liability	13
Document Conventions	14
CHAPTER 1 – CONFIGURATION TEMPLATE BASICS	15
Introducing Configuration Templates	15
Developing Configuration Templates	16
CHAPTER 2 – WORKFLOW TEMPLATE BASICS.....	39
Introducing Workflow Templates	39
Developing Workflow Templates	41
CHAPTER 3 – COMMON VELOCITY TEMPLATE LANGUAGE (VTL) SYNTAX ..	61
Variables	61
Properties	61
Setting Variables	62
Using Logical Expressions	62
Iterating Through a List	62
CHAPTER 4 – DEVICE ADAPTERS.....	63
Device Variables	63
Identifying Bean Classes	66
Using Multiple Devices	68
CLI-based Error Detection	70
CHAPTER 5 – STANDARD AND RADWARE VELOCITY TOOLS.....	73
Velocity Tools in Configuration Templates	73
Radware Velocity Tools in Configuration Templates	73
CHAPTER 6 – CONFIGURATION TEMPLATES DIRECTIVE REFERENCE	85
Answer Directive	85
DefensePro Paste Mode Directive	87
Device Directive	87
DeviceApi Directive	89

Device Lock Directive	89
Error Directive	90
Expect Directive	90
Log Directive	92
Mock Device Directive	92
Parameter Directive	93
Property Directive	95
Result Directive	96
Save Replies Directive	98
Select Directive	98
Timer Directive	99
Typedef Directive	99
 CHAPTER 7 – WORKFLOW TEMPLATE REFERENCE	 103
Workflow Element Definitions	103
Workflow Concepts	103
Velocity Expressions	104
Workflow Descriptor	105
Workflow Actions	107
Inputs	108
Outputs	110
Results	111
Local Devices	112
Sequences	113
Error Handling	113
Parameter Mapping in Steps	114
Including Child Workflows	116
Adding Create and Destroy Actions	117
The \$workflow Local Variable	117
Device Variables	118
DefensePro Devices	120
Workflow Steps	121
Device Steps	121
Child Workflow Steps	125
Scripting Steps	126
Resource Steps	128
Conditional Execution Steps	129
Utility Steps	130
 CHAPTER 8 – INPUT AND OUTPUT PARAMETER TYPES.....	 133
Parameter Type Reference	133

CHAPTER 9 – AUTHORIZING WIZARDS	137
autoName	137
RunAs Sequence	137
<runNext> Element	138
Sequence Termination	139
Navigating Back	141
 RADWARE LTD. END USER LICENSE AGREEMENT	 143

Important Notices

The following important notices are presented in English, French, and German.

Important Notices

This guide is delivered subject to the following conditions and restrictions:

Copyright Radware Ltd. 2019. All rights reserved.

The copyright and all other intellectual property rights and trade secrets included in this guide are owned by Radware Ltd.

The guide is provided to Radware customers for the sole purpose of obtaining information with respect to the installation and use of the Radware products described in this document, and may not be used for any other purpose.

The information contained in this guide is proprietary to Radware and must be kept in strict confidence.

It is strictly forbidden to copy, duplicate, reproduce or disclose this guide or any part thereof without the prior written consent of Radware.

Notice importante

Ce guide est sujet aux conditions et restrictions suivantes:

Copyright Radware Ltd. 2019. Tous droits réservés.

Le copyright ainsi que tout autre droit lié à la propriété intellectuelle et aux secrets industriels contenus dans ce guide sont la propriété de Radware Ltd.

Ce guide d'informations est fourni à nos clients dans le cadre de l'installation et de l'usage des produits de Radware décrits dans ce document et ne pourra être utilisé dans un but autre que celui pour lequel il a été conçu.

Les informations répertoriées dans ce document restent la propriété de Radware et doivent être conservées de manière confidentielle.

Il est strictement interdit de copier, reproduire ou divulguer des informations contenues dans ce manuel sans avoir obtenu le consentement préalable écrit de Radware.

Wichtige Anmerkung

Dieses Handbuch wird vorbehaltlich folgender Bedingungen und Einschränkungen ausgeliefert:

Copyright Radware Ltd. 2019. Alle Rechte vorbehalten.

Das Urheberrecht und alle anderen in diesem Handbuch enthaltenen Eigentumsrechte und Geschäftsgeheimnisse sind Eigentum von Radware Ltd.

Dieses Handbuch wird Kunden von Radware mit dem ausschließlichen Zweck ausgehändigt, Informationen zu Montage und Benutzung der in diesem Dokument beschriebene Produkte von Radware bereitzustellen. Es darf für keinen anderen Zweck verwendet werden.

Die in diesem Handbuch enthaltenen Informationen sind Eigentum von Radware und müssen streng vertraulich behandelt werden.

Es ist streng verboten, dieses Handbuch oder Teile daraus ohne vorherige schriftliche Zustimmung von Radware zu kopieren, vervielfältigen, reproduzieren oder offen zu legen.

Copyright Notices

The following copyright notices are presented in English, French, and German.

Copyright Notices

The programs included in this product are subject to a restricted use license and can only be used in conjunction with this application.

This product contains the Rijndael cipher

The Rijndael implementation by Vincent Rijmen, Antoon Bosselaers and Paulo Barreto is in the public domain and distributed with the following license:

@version 3.0 (December 2000)

Optimized ANSI C code for the Rijndael cipher (now AES)

@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>

@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>

@author Paulo Barreto <paulo.barreto@terra.com.br>

The OnDemand Switch may use software components licensed under the GNU General Public License Agreement Version 2 (GPL v.2) including LinuxBios and Filo open source projects. The source code of the LinuxBios and Filo is available from Radware upon request. A copy of the license can be viewed at: <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>.

This code is hereby placed in the public domain.

This product contains code developed by the OpenBSD Project

Copyright ©1983, 1990, 1992, 1993, 1995

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This product includes software developed by Markus Friedl

This product includes software developed by Theo de Raadt

This product includes software developed by Niels Provos

This product includes software developed by Dug Song

This product includes software developed by Aaron Campbell

This product includes software developed by Damien Miller

This product includes software developed by Kevin Steves

This product includes software developed by Daniel Kouril

This product includes software developed by Wesley Griffin

This product includes software developed by Per Allansson

This product includes software developed by Nils Nordman

This product includes software developed by Simon Wilkinson

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This product contains work derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm. RSA Data Security, Inc. makes no representations concerning either the merchantability of the MD5 Message - Digest Algorithm or the suitability of the MD5 Message - Digest Algorithm for any particular purpose. It is provided "as is" without express or implied warranty of any kind.

Notice traitant du copyright

Les programmes intégrés dans ce produit sont soumis à une licence d'utilisation limitée et ne peuvent être utilisés qu'en lien avec cette application.

Ce produit renferme des codes développés dans le cadre du projet OpenSSL.

Ce produit inclut un logiciel développé dans le cadre du projet OpenSSL. Pour un usage dans la boîte à outils OpenSSL (<http://www.openssl.org/>).

Copyright ©1998-2005 Le projet OpenSSL. Tous droits réservés. Ce produit inclut la catégorie de chiffre Rijndael.

L'implémentation de Rijndael par Vincent Rijmen, Antoon Bosselaers et Paulo Barreto est du domaine public et distribuée sous les termes de la licence suivante:

@version 3.0 (Décembre 2000)

Code ANSI C code pour Rijndael (actuellement AES)

@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>

@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>

@author Paulo Barreto <paulo.barreto@terra.com.br>.

Le commutateur OnDemand peut utiliser les composants logiciels sous licence, en vertu des termes de la licence GNU General Public License Agreement Version 2 (GPL v.2), y compris les projets à source ouverte LinuxBios et Filo. Le code source de LinuxBios et Filo est disponible sur demande auprès de Radware. Une copie de la licence est répertoriée sur: <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>.

Ce code est également placé dans le domaine public.

Ce produit renferme des codes développés dans le cadre du projet OpenSSL.

Copyright ©1983, 1990, 1992, 1993, 1995

Les membres du conseil de l'Université de Californie. Tous droits réservés.

La distribution et l'usage sous une forme source et binaire, avec ou sans modifications, est autorisée pour autant que les conditions suivantes soient remplies:

1. La distribution d'un code source doit inclure la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.
2. La distribution, sous une forme binaire, doit reproduire dans la documentation et/ou dans tout autre matériel fourni la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.
3. Le nom de l'université, ainsi que le nom des contributeurs ne seront en aucun cas utilisés pour approuver ou promouvoir un produit dérivé de ce programme sans l'obtention préalable d'une autorisation écrite.

Ce produit inclut un logiciel développé par Markus Friedl

Ce produit inclut un logiciel développé par Theo de Raadt Ce produit inclut un logiciel développé par Niels Provos

Ce produit inclut un logiciel développé par Dug Song

Ce produit inclut un logiciel développé par Aaron Campbell Ce produit inclut un logiciel développé par Damien Miller

Ce produit inclut un logiciel développé par Kevin Steves

Ce produit inclut un logiciel développé par Daniel Kouril

Ce produit inclut un logiciel développé par Wesley Griffin

Ce produit inclut un logiciel développé par Per Allansson

Ce produit inclut un logiciel développé par Nils Nordman

Ce produit inclut un logiciel développé par Simon Wilkinson.

La distribution et l'usage sous une forme source et binaire, avec ou sans modifications, est autorisée pour autant que les conditions suivantes soient remplies:

1. La distribution d'un code source doit inclure la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant.

La distribution, sous une forme binaire, doit reproduire dans la documentation et/ou dans tout autre matériel fourni la notice de copyright mentionnée ci-dessus, cette liste de conditions et l'avis de non-responsabilité suivant. LE LOGICIEL MENTIONNÉ CI-DESSUS EST FOURNI TEL QUEL PAR LE DÉVELOPPEUR ET TOUTE GARANTIE, EXPLICITE OU IMPLICITE, Y COMPRIS, MAIS SANS S'Y LIMITER, TOUTE GARANTIE IMPLICITE DE QUALITÉ MARCHANDE ET D'ADÉQUATION À UN USAGE PARTICULIER EST EXCLUE.

EN AUCUN CAS L'AUTEUR NE POURRA ÊTRE TENU RESPONSABLE DES DOMMAGES DIRECTS, INDIRECTS, ACCESSOIRES, SPÉCIAUX, EXEMPLAIRES OU CONSÉCUTIFS (Y COMPRIS, MAIS SANS S'Y LIMITER, L'ACQUISITION DE BIENS OU DE SERVICES DE REMPLACEMENT, LA PERTE D'USAGE, DE DONNÉES OU DE PROFITS OU L'INTERRUPTION DES AFFAIRES), QUELLE QU'EN SOIT LA CAUSE ET LA THÉORIE DE RESPONSABILITÉ, QU'IL S'AGISSE D'UN CONTRAT, DE RESPONSABILITÉ STRICTE OU D'UN ACTE DOMMAGEABLE (Y COMPRIS LA NÉGLIGENCE OU AUTRE), DÉCOULANT DE QUELLE QUE FAÇON QUE CE SOIT DE L'USAGE DE CE LOGICIEL, MÊME S'IL A ÉTÉ AVERTI DE LA POSSIBILITÉ D'UN TEL DOMMAGE.

Copyrightvermerke

Die in diesem Produkt enthalten Programme unterliegen einer eingeschränkten Nutzungslizenz und können nur in Verbindung mit dieser Anwendung benutzt werden.

Dieses Produkt enthält einen vom OpenSSL-Projekt entwickelten Code.

Dieses Produkt enthält vom OpenSSL-Projekt entwickelte Software. Zur Verwendung im OpenSSL Toolkit (<http://www.openssl.org/>).

Copyright ©1998-2005 The OpenSSL Project. Alle Rechte vorbehalten. Dieses Produkt enthält die Rijndael cipher

Die Rijndael-Implementierung von Vincent Rijndael, Anton Bosselaers und Paulo Barreto ist öffentlich zugänglich und wird unter folgender Lizenz vertrieben:

@version 3.0 (December 2000)

Optimierter ANSI C Code für den Rijndael cipher (jetzt AES)

@author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>

@author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>

@author Paulo Barreto <paulo.barreto@terra.com.br>

Der OnDemand Switch verwendet möglicherweise Software, die im Rahmen der DNU Allgemeine Öffentliche Lizenzvereinbarung Version 2 (GPL v.2) lizenziert sind, einschließlich LinuxBios und Filo Open Source-Projekte. Der Quellcode von LinuxBios und Filo ist bei Radware auf Anfrage erhältlich. Eine Kopie dieser Lizenz kann eingesehen werden unter:

<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

Dieser Code wird hiermit allgemein zugänglich gemacht.

Dieses Produkt enthält einen vom OpenBSD-Projekt entwickelten Code

Copyright ©1983, 1990, 1992, 1993, 1995

The Regents of the University of California. Alle Rechte vorbehalten.

Die Verbreitung und Verwendung in Quell- und binärem Format, mit oder ohne Veränderungen, sind unter folgenden Bedingungen erlaubt:

1. Die Verbreitung von Quellcodes muss den voranstehenden Copyrightvermerk, diese Liste von Bedingungen und den folgenden Haftungsausschluss beibehalten.
2. Die Verbreitung in binärem Format muss den voranstehenden Copyrightvermerk, diese Liste von Bedingungen und den folgenden Haftungsausschluss in der Dokumentation und/oder andere Materialien, die mit verteilt werden, reproduzieren.
3. Weder der Name der Universität noch die Namen der Beitragenden dürfen ohne ausdrückliche vorherige schriftliche Genehmigung verwendet werden, um von dieser Software abgeleitete Produkte zu empfehlen oder zu bewerben.

Dieses Produkt enthält von Markus Friedl entwickelte Software

Dieses Produkt enthält von Theo de Raadt entwickelte Software

Dieses Produkt enthält von Niels Provos entwickelte Software

Dieses Produkt enthält von Dug Song entwickelte Software

Dieses Produkt enthält von Aaron Campbell entwickelte Software

Dieses Produkt enthält von Damien Miller entwickelte Software

Dieses Produkt enthält von Kevin Steves entwickelte Software

Dieses Produkt enthält von Daniel Kouril entwickelte Software

Dieses Produkt enthält von Wesley Griffin entwickelte Software

Dieses Produkt enthält von Per Allansson entwickelte Software

Dieses Produkt enthält von Nils Nordman entwickelte Software

Dieses Produkt enthält von Simon Wilkinson entwickelte Software

Die Verbreitung und Verwendung in Quell- und binärem Format, mit oder ohne Veränderungen, sind unter folgenden Bedingungen erlaubt:

1. Die Verbreitung von Quellcodes muss den voranstehenden Copyrightvermerk, diese Liste von Bedingungen und den folgenden Haftungsausschluss beibehalten.

Die Verbreitung in binärem Format muss den voranstehenden Copyrightvermerk, diese Liste von Bedingungen und den folgenden Haftungsausschluss in der Dokumentation und/oder andere Materialien, die mit verteilt werden, reproduzieren. SÄMTLICHE VORGENANNTTE SOFTWARE WIRD VOM AUTOR IM IST-ZUSTAND ("AS IS") BEREITGESTELLT. JEDLICHE AUSDRÜCKLICHEN ODER IMPLIZITEN GARANTIE, EINSCHLIESSLICH, DOCH NICHT BESCHRÄNKT AUF DIE IMPLIZIERTEN GARANTIE DER MARKTGÄNGIGKEIT UND DER ANWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK, SIND AUSGESCHLOSSEN.

UNTER KEINEN UMSTÄNDEN HAFTET DER AUTOR FÜR DIREKTE ODER INDIREKTE SCHÄDEN, FÜR BEI VERTRAGSERFÜLLUNG ENTSTANDENE SCHÄDEN, FÜR BESONDERE SCHÄDEN, FÜR SCHADENSERSATZ MIT STRAFCHARAKTER, ODER FÜR FOLGESCHÄDEN EINSCHLIESSLICH, DOCH NICHT BESCHRÄNKT AUF, ERWERB VON ERSATZGÜTERN ODER ERSATZLEISTUNGEN; VERLUST AN NUTZUNG, DATEN ODER GEWINN; ODER GESCHÄFTSUNTERBRECHUNGEN) GLEICH, WIE SIE ENTSTANDEN SIND, UND FÜR JEDLICHE ART VON HAFTUNG, SEI ES VERTRÄGE, GEFÄHRDUNGSHAFTUNG, ODER DELIKTISCHE HAFTUNG (EINSCHLIESSLICH FAHRLÄSSIGKEIT ODER ANDERE), DIE IN JEDLICHER FORM FOLGE DER BENUTZUNG DIESER SOFTWARE IST, SELBST WENN AUF DIE MÖGLICHKEIT EINES SOLCHEN SCHADENS HINGEWIESEN WURDE.

Standard Warranty

The following standard warranty is presented in English, French, and German.

Standard Warranty

Radware offers a limited warranty for all its products ("Products"). Radware hardware products are warranted against defects in material and workmanship for a period of one year from date of shipment. Radware software carries a standard warranty that provides bug fixes for up to 90 days after date of purchase. Should a Product unit fail anytime during the said period(s), Radware will, at its discretion, repair or replace the Product.

For hardware warranty service or repair, the product must be returned to a service facility designated by Radware. Customer shall pay the shipping charges to Radware and Radware shall pay the shipping charges in returning the product to the customer. Please see specific details outlined in the Standard Warranty section of the customer's purchase order.

Radware shall be released from all obligations under its Standard Warranty in the event that the Product and/or the defective component has been subjected to misuse, neglect, accident or improper installation, or if repairs or modifications were made by persons other than Radware authorized service personnel, unless such repairs by others were made with the written consent of Radware.

EXCEPT AS SET FORTH ABOVE, ALL RADWARE PRODUCTS (HARDWARE AND SOFTWARE) ARE PROVIDED BY "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.

Garantie standard

Radware octroie une garantie limitée pour l'ensemble de ses produits (" Produits "). Le matériel informatique (hardware) Radware est garanti contre tout défaut matériel et de fabrication pendant une durée d'un an à compter de la date d'expédition. Les logiciels (software) Radware sont fournis avec une garantie standard consistant en la fourniture de correctifs des dysfonctionnements du logiciels (bugs) pendant une durée maximum de 90 jours à compter de la date d'achat. Dans l'hypothèse où un Produit présenterait un défaut pendant ladite(lesdites) période(s), Radware procédera, à sa discrétion, à la réparation ou à l'échange du Produit.

S'agissant de la garantie d'échange ou de réparation du matériel informatique, le Produit doit être retourné chez un réparateur désigné par Radware. Le Client aura à sa charge les frais d'envoi du Produit à Radware et Radware supportera les frais de retour du Produit au client. Veuillez consulter les conditions spécifiques décrites dans la partie " Garantie Standard " du bon de commande client.

Radware est libérée de toutes obligations liées à la Garantie Standard dans l'hypothèse où le Produit et/ou le composant défectueux a fait l'objet d'un mauvais usage, d'une négligence, d'un accident ou d'une installation non conforme, ou si les réparations ou les modifications qu'il a subi ont été effectuées par d'autres personnes que le personnel de maintenance autorisé par Radware, sauf si Radware a donné son consentement écrit à ce que de telles réparations soient effectuées par ces personnes.

SAUF DANS LES CAS PREVUS CI-DESSUS, L'ENSEMBLE DES PRODUITS RADWARE (MATERIELS ET LOGICIELS) SONT FOURNIS " TELS QUELS " ET TOUTES GARANTIES EXPRESSES OU IMPLICITES SONT EXCLUES, EN CE COMPRIS, MAIS SANS S'Y RESTREINDRE, LES GARANTIES IMPLICITES DE QUALITE MARCHANDE ET D'ADEQUATION A UNE UTILISATION PARTICULIERE.

Standard Garantie

Radware bietet eine begrenzte Garantie für alle seine Produkte ("Produkte") an. Hardware Produkte von Radware haben eine Garantie gegen Material- und Verarbeitungsfehler für einen Zeitraum von einem Jahr ab Lieferdatum. Radware Software verfügt über eine Standard Garantie zur Fehlerbereinigung für einen Zeitraum von bis zu 90 Tagen nach Erwerbsdatum. Sollte ein Produkt innerhalb des angegebenen Garantiezeitraumes einen Defekt aufweisen, wird Radware das Produkt nach eigenem Ermessen entweder reparieren oder ersetzen.

Für den Hardware Garantieservice oder die Reparatur ist das Produkt an eine von Radware bezeichnete Serviceeinrichtung zurückzugeben. Der Kunde hat die Versandkosten für den Transport des Produktes zu Radware zu tragen, Radware übernimmt die Kosten der Rückversendung des Produktes an den Kunden. Genauere Angaben entnehmen Sie bitte dem Abschnitt zur Standard Garantie im Bestellformular für Kunden.

Radware ist von sämtlichen Verpflichtungen unter seiner Standard Garantie befreit, sofern das Produkt oder der fehlerhafte Teil zweckentfremdet genutzt, in der Pflege vernachlässigt, einem Unfall ausgesetzt oder unsachgemäß installiert wurde oder sofern Reparaturen oder Modifikationen von anderen Personen als durch Radware autorisierten Kundendienstmitarbeitern vorgenommen wurden, es sei denn, diese Reparatur durch besagte andere Personen wurden mit schriftlicher Genehmigung seitens Radware durchgeführt.

MIT AUSNAHME DES OBEN DARGESTELLTEN, SIND ALLE RADWARE PRODUKTE (HARDWARE UND SOFTWARE) GELIEFERT "WIE GESEHEN" UND JEDLICHE AUSDRÜCKLICHEN ODER STILLSCHWEIGENDEN GARANTIEN, EINSCHLIESSLICH ABER NICHT BEGRENZT AUF STILLSCHWEIGENDE GEWÄHRLEISTUNG DER MARKTFÄHIGKEIT UND EIGNUNG FÜR EINEN BESTIMMTEN ZWECK AUSGESCHLOSSEN.

Limitations on Warranty and Liability

The following limitations on warranty and liability are presented in English, French, and German.

Limitations on Warranty and Liability

IN NO EVENT SHALL RADWARE LTD. OR ANY OF ITS AFFILIATED ENTITIES BE LIABLE FOR ANY DAMAGES INCURRED BY THE USE OF THE PRODUCTS (INCLUDING BOTH HARDWARE AND SOFTWARE) DESCRIBED IN THIS USER GUIDE, OR BY ANY DEFECT OR INACCURACY IN THIS USER GUIDE ITSELF. THIS INCLUDES BUT IS NOT LIMITED TO ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION). THE ABOVE LIMITATIONS WILL APPLY EVEN IF RADWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

Limitations de la Garantie et Responsabilité







RADWARE LTD. OU SES ENTITIES AFFILIES NE POURRONT EN AUCUN CAS ETRE TENUES RESPONSABLES DES DOMMAGES SUBIS DU FAIT DE L'UTILISATION DES PRODUITS (EN CE COMPRIS LES MATERIELS ET LES LOGICIELS) DECRITS DANS CE MANUEL D'UTILISATION, OU DU FAIT DE DEFAUT OU D'IMPRECISIONS DANS CE MANUEL D'UTILISATION, EN CE COMPRIS, SANS TOUTEFOIS QUE CETTE ENUMERATION SOIT CONSIDEREE COMME LIMITATIVE, TOUS DOMMAGES DIRECTS, INDIRECTS, ACCIDENTELS, SPECIAUX, EXEMPLAIRES, OU ACCESSOIRES (INCLUANT, MAIS SANS S'Y RESTREINDRE, LA FOURNITURE DE PRODUITS OU DE SERVICES DE REMPLACEMENT; LA PERTE D'UTILISATION, DE DONNEES OU DE PROFITS; OU L'INTERRUPTION DES AFFAIRES). LES LIMITATIONS CI-DESSUS S'APPLIQUERONT QUAND BIEN MEME RADWARE A ETE INFORMEE DE LA POSSIBLE EXISTENCE DE CES DOMMAGES. CERTAINES JURIDICTIONS N'ADMETTANT PAS LES EXCLUSIONS OU LIMITATIONS DE GARANTIES IMPLICITES OU DE RESPONSABILITE EN CAS DE DOMMAGES ACCESSOIRES OU INDIRECTS, LESDITES LIMITATIONS OU EXCLUSIONS POURRAIENT NE PAS ETRE APPLICABLE DANS VOTRE CAS.

Haftungs- und Gewährleistungsausschluss

IN KEINEM FALL IST RADWARE LTD. ODER EIN IHR VERBUNDENES UNTERNEHMEN HAFTBAR FÜR SCHÄDEN, WELCHE BEIM GEBRAUCH DES PRODUKTES (HARDWARE UND SOFTWARE) WIE IM BENUTZERHANDBUCH BESCHRIEBEN, ODER AUFGRUND EINES FEHLERS ODER EINER UNGENAUIGKEIT IN DIESEM BENUTZERHANDBUCH SELBST ENTSTANDEN SIND. DAZU GEHÖREN UNTER ANDEREM (OHNE DARAUF BEGRENZT ZU SEIN) JEDGLICHE DIREKTEN; IDIREKTEN; NEBEN; SPEZIELLEN, BELEGTEN ODER FOLGESCHÄDEN (EINSCHLIESSLICH ABER NICHT BEGRENZT AUF BESCHAFFUNG ODER ERSATZ VON WAREN ODER DIENSTEN, NUTZUNGSAusFALL, DATEN- ODER GEWINNVERLUST ODER BETRIEBSUNTERBRECHUNGEN). DIE OBEN GENANNTE BEGRENZUNGEN GREIFEN AUCH, SOFERN RADWARE AUF DIE MÖGLICHKEIT EINES SOLCHEN SCHADENS HINGEWIESEN WORDEN SEIN SOLLTE. EINIGE RECHTSORDNUNGEN LASSEN EINEN AUSSCHLUSS ODER EINE BEGRENZUNG STILLSCHWEIGENDER GARANTIE ODER HAFTUNGEN BEZÜGLICH NEBEN- ODER FOLGESCHÄDEN NICHT ZU, SO DASS DIE OBEN DARGESTELLTE BEGRENZUNG ODER DER AUSSCHLUSS SIE UNTER UMSTÄNDEN NICHT BETREFFEN WIRD.

Document Conventions

The following describes the conventions and symbols that this guide uses:

Item	Description	Description	Beschreibung
 Example	An example scenario	Un scénario d'exemple	Ein Beispielszenarium
 Caution:	Possible damage to equipment, software, or data	Endommagement possible de l'équipement, des données ou du logiciel	Mögliche Schäden an Gerät, Software oder Daten
 Note:	Additional information	Informations complémentaires	Zusätzliche Informationen
 To	A statement and instructions	Références et instructions	Eine Erklärung und Anweisungen
 Tip:	A suggestion or workaround	Une suggestion ou solution	Ein Vorschlag oder eine Umgehung
 Warning:	Possible physical harm to the operator	Blessure possible de l'opérateur	Verletzungsgefahr des Bedieners

CHAPTER 1 – CONFIGURATION TEMPLATE BASICS

This section describes the following topics:

- [Introducing Configuration Templates, page 15](#)
- [Developing Configuration Templates, page 16](#)

Introducing Configuration Templates

Configuration templates are a powerful method of automating tedious and error-prone configuration tasks. Configuration templates are applicable to every Radware device that exposes a CLI interface. The value of configuration templates is best illustrated through the following examples:



Examples

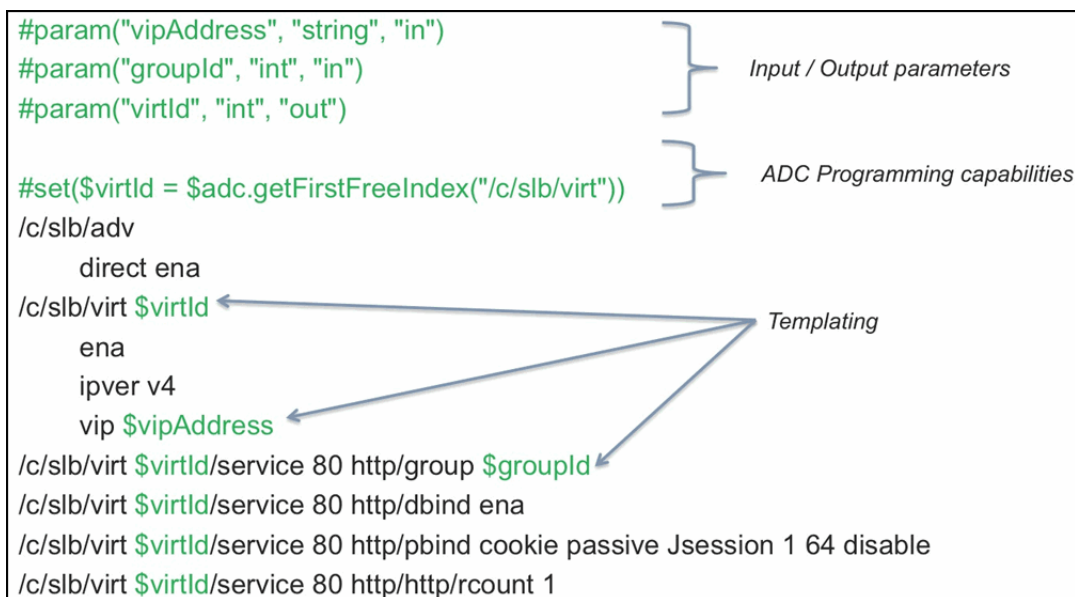
- A** Imagine an administrator who is on-boarding 20 new Alteon VX ADCs, all sharing the same basic configuration (SNMP, SMTP, NTP, CLI, SSH, SysLog, and so on). In the current mode of operation, the administrator would have to manually configure each ADC and repeat the configuration task over and over, until all ADCs are configured. It is easy to see that this mode of operation is obviously time-consuming, tedious and error-prone. With vDirect configuration templates, this task can be made very easy, fast and error-proof.
- B** Now the administrator has to ensure that all of the organization's SSL certificates, serving the organization's applications, are valid at all times. The administrator has to verify this across dozens of ADCs. As with the first example, doing this manually is a tedious, time-consuming and error-prone operation. Here too, vDirect configuration templates can make this task extremely easy, fast and error-proof.

Benefits

Users who are familiar with the CLI of Radware devices can harness these skills into writing configuration templates once and running them multiple times. vDirect enhances the standard Alteon CLI by adding a proven and widely used template engine that allows for the addition of input/output parameters and light-programming capabilities. The immediate benefits of vDirect configuration templates are:

- **Save time**—Write the CLI-based configuration template once and run it multiple times with few changes to ADC-specific parameters.
- **Error-Free**—Writing the CLI code into the template once only reduces the risk of creating erroneous CLI commands while configuring multiple ADCs.
- **Free up expensive IT resources**—Let your ADC and networking professionals develop complex configuration templates once, while allowing less proficient IT personnel to take advantage of those templates and integrate the ADC services as part of the applications they provision.

Figure 1: vDirect Configuration Template Example



Developing Configuration Templates

Configuration templates allow users to perform multi-step manual configuration tasks in simple repeatable units available through the vDirect API, vDirect Web interface, or Vision Operator Tool Box.

In this section, you will learn the fundamentals of writing configuration templates, how to parameterize input/output data, and how to deploy and run configuration templates.

This section describes the following topics:

- [Prerequisites, page 17](#)
- [Configuration Template Development Steps, page 17](#)
- [Preparing Your Environment, page 18](#)
- [Managing Configuration Templates, page 18](#)
- [Example and Exercises—Creating a Real Server, page 22](#)
- [Example and Exercises—Creating a Virtual Server, page 27](#)
- [Example and Exercises—Deleting a Virtual Server, page 33](#)
- [Troubleshooting Configuration Templates, page 37](#)

Prerequisites

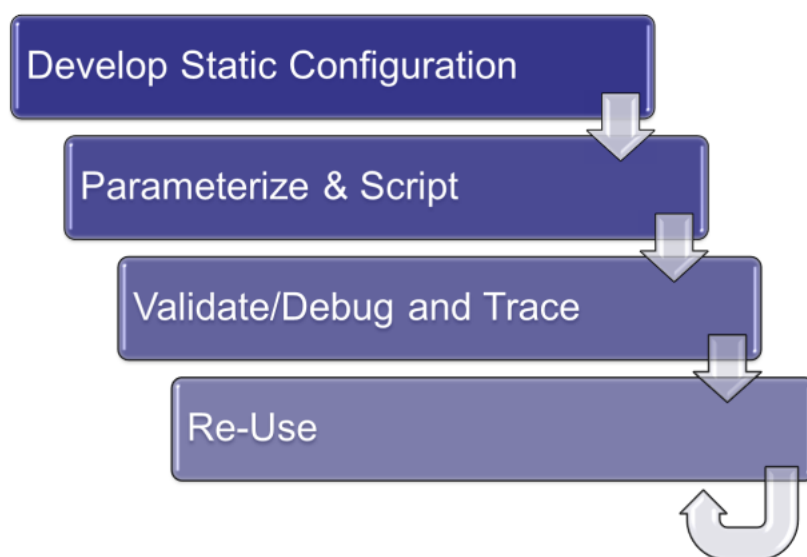
The following prerequisites are necessary for developing configuration templates:

- At least one Alteon (standalone, vADC/VX or virtual) configured for access from SSH and SNMP/ Web API. Radware recommends Alteon version 30.2.1.0 or later.
- At least one vDirect VA connected to a network for accessing the Alteon instance.

Configuration Template Development Steps

Radware recommends that you follow the steps shown in [Figure 2 - Recommended Configuration Template Development Steps, page 17](#) when developing a configuration template.

Figure 2: Recommended Configuration Template Development Steps



1. **Develop Static Configuration**—Verify that you have a working device configuration that achieves the results you require.
2. **Parameterize & Script**—Use the appropriate sections from a working configuration dump to identify the following:
 - The parameters that are mandatory for the configuration, and the parameters that must always be passed by the template. The result defines the input and output parameters of the configuration template.
 - The sections of the Alteon configuration that must be available to vDirect (for use as Java beans).
 - The sections of the configuration that are repeatable blocks. The result defines which sections of the configuration template to design in an iterative way.
3. **Validate/Debug and Trace**—Use the vDirect Web interface to test the configuration template until it generates the required results. Test the configuration template on different Alteon form factors as necessary.
4. **Re-Use**—You now have a configuration template that is appropriate for use in different circumstances and on different devices.

Preparing Your Environment

This section describes how to set up vDirect and connect it to Alteon, and introduces some aspects of the vDirect Web interface.

This section describes the following topics:

- [Logging In to vDirect, page 18](#)
- [Registering Alteon, page 18](#)

Logging In to vDirect

This section describes how to log in to the vDirect Web interface.



To log in to the vDirect Web interface

1. In your Internet browser, enter <https://<vdirect-ip-or-hostname>:2189/ui>.
2. Log in using the following credentials:
 - user name: vDirect
 - password: radware

The vDirect configuration user interface displays.

Registering Alteon

This section describes how to register a dedicated Alteon with vDirect.



To register a dedicated Alteon with vDirect

1. Select **Inventory > ADC Containers**.
2. Click **Register** and select **Alteon Dedicated**.
The **Register Alteon Dedicated Container** dialog box displays.
3. Set the following parameters as shown:
 - **Address**—The management IP address of the Alteon.
 - **Name**—Alteon-01.
 - **Configuration Access Protocol**—Select SNMP or HTTPS based on the way you have configured your Alteon device to be managed.
4. Configure the rest of the parameters based on your Alteon configuration.
5. When all parameters are correct, click **Register**.

If the configuration is valid, the dialog box closes and your Alteon displays in the table.

Managing Configuration Templates

This section introduces some aspects of the configuration template capabilities of the vDirect Web interface.

This section describes the following topics:

- [Viewing Configuration Templates, page 19](#)
- [Creating Your First Configuration Template, page 19](#)
- [Dry Running Your First Configuration Template, page 20](#)

- [Running Your First Configuration Template, page 20](#)
- [Parameterizing Your First Configuration Template, page 21](#)

Viewing Configuration Templates

This section describes how to view configuration templates that already exist in vDirect.



To view configuration templates

- > Select **Inventory > Configuration Templates**.

A table listing all the configuration templates defined in vDirect displays. If this is your first use, the table is empty.

Creating Your First Configuration Template

This section describes how to create your first configuration template.



To create your first configuration template

1. Select **Inventory > Configuration Templates**.
2. Click **New**.

The **New Template** dialog box displays.

3. In the **Name** field, type **idle999.vm**.
4. Click in the text area below the **Tenants** option and type `/c/sys/idle 999`.
5. Click **Save**.

If the template syntax is valid, the **New Template** dialog box closes and **idle99.vm** displays in the list of configuration templates.

Dry Running Your First Configuration Template

Dry running a template allows you to see only the expected generated CLI commands while running the template.



To dry run a configuration template

1. Select **Inventory > Configuration Templates**.
2. In the table, select the checkbox next to the **idle999.vm** template and click **Run**.

The **Run Configuration Template** screen displays.

In the **Devices** section you can select one or more Alteons for configuration. If the template does not specify an Alteon, vDirect uses the default **adc** Alteon.

The **Parameters** section remains empty since you have not yet defined parameters for the **idle999.vm** template.

3. Select **Dry Run** to show only the expected generated CLI commands while running the template.
4. Select the **Registered Devices** dialog box, and then select the checkbox next to **Alteon-01**.
5. Click **Run**.

The template status screen displays.

6. Select **Generated Script**.

A message similar to the following displays:

```
/* Generated script for device adc@xxx.xxx.xxx.xxx */  
/c/sys/idle 999
```



Note: Dry running a configuration template does not change the Alteon configuration.

7. Click **Close** to close the template status screen.
8. Use the Back button of your browser to return to the **Configuration Templates** screen.

Running Your First Configuration Template

This section describes how to run your first configuration template, and how to use vDirect to apply and save the Alteon configuration.



To run your first configuration template

1. Select **Inventory > Configuration Templates**.
 2. In the table, select the checkbox next to the **idle999.vm** template and click **Run**.
- The **Run Configuration Template** screen displays.
3. Verify that **Dry Run** is not selected.
 4. Select the **Registered Devices** dialog box, and then select the checkbox next to **Alteon-01**.
 5. Click **Run**.

The template status screen displays.

6. Select **CLI Output**.

The CLI transcript of the interaction between vDirect and Alteon displays.

Although vDirect has set the Alteon configuration parameters, the configuration has not yet been applied and saved.

Applying and Saving the Alteon Configuration

Question: How can you modify the **idle999.vm** template to also apply and save the Alteon configuration?



To apply and save the Alteon configuration

1. In template status screen, click **Close** to close the screen.
The template status screen displays.
2. In the text area, add apply and save commands to the code, as follows:

```
/c/sys/idle 999  
apply  
save
```

3. Click **Save**.

Parameterizing Your First Configuration Template

This section describes how to amend the static **idle999.vm** configuration template to a template that accepts the idle time as a parameter.



To amend the idle999.vm configuration template to accept the idle time parameter

1. Select **Inventory > Configuration Templates**.
2. In the table, select the checkbox next to the **idle999.vm** template and click **Download**.
vDirect downloads a file named **idle999.vm** to your computer.
3. Locate the **idle999.vm** file (it is probably in the **Downloads** directory of your computer) and rename it **idle.vm**.
4. In the **Configuration Templates** table, deselect the checkbox next to the **idle999.vm** template.
5. Click **Upload** and browse to the **idle.vm** file.
6. Select the **idle.vm** file and click **Open**.
vDirect adds the **idle.vm** template to the **Configuration Templates** table.
7. Click the [idle.vm](#) link to edit it.
The **Configuration Template** screen displays.
8. In the text area, add an **\$idle_time** parameter as shown below:

```
#param($idle_time, 'type=int', 'direction=in', 'prompt=Idle Time  
(Seconds)', 'min=1', 'max=10080')  
  
/c/sys/idle $idle_time  
apply  
save
```

9. Click **Save** to close the **Configuration Template** screen.
10. Use the Back button of your browser to return to the **Configuration Templates** screen.
11. In the table, select the checkbox next to the **idle999.vm** template and click **Run**.
The **Run Configuration Template** screen displays.
12. Select the **Registered Devices** dialog box, and then select the checkbox next to **Alteon-01**.
13. In the **Idle Time (Seconds)** parameter, type a number between 1 and 10080.
14. Click **Run**.
The template status screen displays.
15. Select **CLI Output**.
The CLI transcript of the interaction between vDirect and Alteon displays.
16. In the generated script and in the CLI output, verify that the idle time value you entered has substituted \$idle_time in the /cfg/sys/idle command that vDirect sends to Alteon.

Example and Exercises—Creating a Real Server

Goal: To create a group with real servers.

Develop Static Configuration

Create a working Alteon configuration with virtual servers, virtual services, groups, and real servers. An example of an Alteon configuration dump can be found in the **Alteon_dump.txt** file.

The section of the configuration that handles the real server configuration and the group configuration is shown below:

```
/c/slb/real real-server-1
    ena
    ipver v4
    rip 192.168.12.10
    name "192.168.12.10"
/c/slb/real real-server-2
    ena
    ipver v4
    rip 192.168.12.11
    name "192.168.12.11"
/c/slb/real real-server-3
    ena
    ipver v4
    rip 192.168.12.12
    name "192.168.12.12"
/c/slb/group vDirect-servers-group
    ipver v4
    add real-server-1
    add real-server-2
    add real-server-3
    name "Group vDirect-servers-group"
```

Question: What are the aspects of this configuration that prevent it from being used multiple times?

Answer:

- The quantity of real servers needed may change.
- The IP address of each real server may be different and not static.
- The real server identifier must be unique for each real server and should not exist before using the configuration template.
- The group identifier must be unique and should not exist before using the configuration template.
- The group must include the appropriate real server identifiers as defined by the specific configuration template usage.

Parameterize & Script

This section describes how to turn the static text above into a configuration template.

Step 1—Create a configuration template with static configuration

1. Log in into the vDirect Web user interface, as described at [Logging In to vDirect, page 18](#).
2. Create a new configuration template and name it **create_group.vm**, as described at [Creating Your First Configuration Template, page 19](#).
3. Paste the section of the configuration that handles the creation of the real servers and group into the **create_group.vm** text and save.
4. Before you run the **create_group.vm** template, select an Alteon. The template issues the CLI commands as specified.

Step 2—Parameterize

In this section we define the parameters that will be requested from the end user as follows:

- **base_name**—A base name that will be used to generate the group and real server identification.
- **server_ips**—An array of IP addresses for the real servers.

Parameters for a configuration template are defined using the **#param** directive. [Configuration Templates Directive Reference, page 85](#) describes directives including the **#param** directive.

Task: Add the **base_name** and **server_ips** parameters to the **create_group.vm** template.

Solution:

1. In the **Configuration Templates** table, click the [create_group.vm](#) link to edit the template.
2. Add the following as the first two lines:

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')
```



Note: Run the **create_group.vm** template and verify that vDirect requests the **base_name** and **server_ips** parameters.

Your template should now look like this:

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

/c/slb/real real-server-1
    ena
    ipver v4
    rip 192.168.12.10
    name "192.168.12.10"
/c/slb/real real-server-2
    ena
    ipver v4
    rip 192.168.12.11
    name "192.168.12.11"
/c/slb/real real-server-3
    ena
    ipver v4
    rip 192.168.12.12
    name "192.168.12.12"
/c/slb/group vDirect-servers-group
    ipver v4
    add real-server-1
    add real-server-2
    add real-server-3
    name "Group vDirect-servers-group"
```


Step 3—Script

In this section we modify the static configuration to use the parameters.

Task 1: Use the **`$base_name`** variable to generate identifiers instead of the static names.

Solution:

1. We use the convention of **`${base_name}_group`** for the group identifier and **`${base_name}_server_ip`** for the server identifier.
2. Modify the **`create_group.vm`** template as follows. Do not modify the **`#param`** lines.

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

/c/slb/real ${base_name}_192_168_12_10
    ena
    ipver v4
    rip 192.168.12.10
    name "192.168.12.10"
/c/slb/real ${base_name}_192_168_12_11
    ena
    ipver v4
    rip 192.168.12.11
    name "192.168.12.11"
/c/slb/real ${base_name}_192_168_12_13
    ena
    ipver v4
    rip 192.168.12.12
    name "192.168.12.12"
/c/slb/group ${base_name}_group
    ipver v4
    add ${base_name}_192_168_12_10
    add ${base_name}_192_168_12_11
    add ${base_name}_192_168_12_12
    name "${base_name} Group"
```



Notes

- Alteon identifiers do not contain the "." character. Replace "." with a valid character for identifiers. For example, "_".
- Since the velocity tool renders the **`$base_name`** as part of the text, syntax such as **`${base_name}_192_168_12_10`** is valid.

Task 2: Iterate over the **`$server_ips`** array to generate the server and group configuration dynamically.

Solution:

1. We use the velocity directive **`#foreach($server_ip in $server_ips)`** to iterate over the array.
2. Since velocity strings are Java strings, all the functions available on a Java string are also available in velocity. Specifically, we use the **`replace('.', '_')`** Java string function to generate a valid identifier from an IP address.

For more information about the **`replace`** Java string function and all other available string functions, see [https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replace\(java.lang.CharSequence,%20java.lang.CharSequence\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#replace(java.lang.CharSequence,%20java.lang.CharSequence)).

3. The group configuration can be performed during a single iteration which modifies the generated CLI sequence from the original form. An alternative (not presented here) that iterates first to generate the servers configuration and then iterates while generating the group configuration would have preserved the original form.
4. Modify the **create_group.vm** template as follows:

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

#set($group_id = "${base_name}_group")

/c/slb/group $group_id
    ipver v4
    name "$base_name Group"

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    /c/slb/real $server_id
        ena
        ipver v4
        rip $server_ip
        name "$server_ip"

    /c/slb/group $group_id
        add $server_id
#end
```

Task 3: Fail if the servers or group existed before running the template.

Solution:

1. To read information from Alteon we use the methods of the **\$adc** variable. This is a special variable, called a Device Adapter, that represents the Alteon and can be used to query Alteon while the template is running. [Device Adapters, page 63](#) describes the **\$adc** functions.
2. To read configuration values from Alteon, specific beans (beans are simply a Java object with specially named methods) need to be used. [Device Adapters, page 63](#) describes how to identify bean classes. In addition to get the full list of available beans, please consult the vDirect Java docs.
3. Alteon version 29.0 introduced default alphanumeric identifiers for the server load balancing configuration. Since beans that map to the numeric configuration are different than beans that map to the alphanumeric configuration, add the prefix **/enh** to the standard CLI path.
4. To check if a group exists, call **\$adc.readBean('/enh/c/slb/group', \$group_id)**.
5. To check if a real server exists, call **\$adc.readBean('/enh/c/slb/real', \$server_id)**.
6. To stop execution and return an error, use the **#error** directive.
7. Modify the **create_group.vm** template as follows:

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

#set($group_id = "${base_name}_group")

## Validation part
#set($found_group = $adc.readBean('enh/c/slb/group', $group_id))
#if($adc.isEmpty($found_group))
    #error("Group $group_id already exists!")
#end

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    #set($found_server = $adc.readBean('enh/c/slb/real', $server_id))
    #if($adc.isEmpty($found_server))
        #error("Server $server_id already exists!")
    #end
#end

## Configuration part
/c/slb/group $group_id
    ipver v4
    name "$base_name Group"

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    /c/slb/real $server_id
        ena
        ipver v4
        rip $server_ip
        name "$server_ip"

    /c/slb/group $group_id
        add $server_id
#end
```

Example and Exercises—Creating a Virtual Server

Goal: To create a simple HTTP virtual server configuration.

Develop Static Configuration

Create a working Alteon configuration with virtual servers, virtual services, groups, and real servers. An example of an Alteon configuration dump can be found in the **Alteon_dump.txt** file.

The section of the configuration that handles the configuration of the HTTP virtual server is shown below:

```
/c/slb/real real-server-1
    ena
    ipver v4
    rip 192.168.12.10
    name "192.168.12.10"
/c/slb/real real-server-2
    ena
    ipver v4
    rip 192.168.12.11
    name "192.168.12.11"
/c/slb/real real-server-3
    ena
    ipver v4
    rip 192.168.12.12
    name "192.168.12.12"
/c/slb/group vDirect-servers-group
    ipver v4
    add real-server-1
    add real-server-2
    add real-server-3
    name "Group vDirect-servers-group"
/c/slb/virt vDirect-virtual-service
    ena
    ipver v4
    vip 192.168.13.190
    vname "vname vDirect-virtual-service"
/c/slb/virt vDirect-virtual-service/service 80 http
    group vDirect-servers-group
    rport 80
```



Note: Automating the creation of the group and real servers is described at [Example and Exercises—Creating a Real Server, page 22](#). This section uses the **create_group.vm** configuration template as a starting point.

Question: What are the aspects of this configuration that prevent it from being used multiple times?

Answer:

1. The IP address of the virtual server may be different and not static.
2. The virtual server identifier must be unique and should not exist before using the configuration template.
3. The group identifier associated with the virtual HTTP service must not be static but refer to the dynamically generated group identifier of the script.

Parameterize & Script

This section describes how to turn the static text above into a configuration template.

Step 1—Create a configuration template with static configuration

1. Verify that you have the **create_group.vm** configuration template (either from [Example and Exercises—Creating a Real Server, page 22](#) or from the files accompanying this document).
2. Copy the **create_group.vm** template to a file named **create_vip.vm**.

3. Log into the vDirect Web user interface, as described at [Logging In to vDirect, page 18](#).
4. Upload the **create_vip.vm** template via the **Configuration Templates** screen.
5. In the **Configuration Templates** table, click the [create_vip.vm](#) link to edit the template.
6. Paste the highlighted section of the configuration that handles the creation of the virtual server below the section that validates and creates the group and real servers, and save.

The result does not generate a valid configuration since it includes the dynamic portion from the previous section with the new static portion that does not yet refer to the dynamic portion.

```
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

## Validation part
#set($group_id = "${base_name}_group")
#set($found_group = $adc.readBean('enh/c/slb/group', $group_id))
#if($adc.isEmpty($found_group))
    #error("Group $group_id already exists!")
#end

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    #set($found_server = $adc.readBean('enh/c/slb/real', $server_id))
    #if($adc.isEmpty($found_server))
        #error("Server $server_id already exists!")
    #end
#end

## Configuration part
/c/slb/group $group_id
    ipver v4
    name "$base_name Group"

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    /c/slb/real $server_id
        ena
        ipver v4
        rip $server_ip
        name "$server_ip"

    /c/slb/group $group_id
        add $server_id
#end

/c/slb/virt vDirect-virtual-service
    ena
    ipver v4
    vip 192.168.13.190
    vname "vname vDirect-virtual-service"

/c/slb/virt vDirect-virtual-service/service 80 http
    group $group_id
    rport 80
```

Step 2—Parameterize

In this section we define the additional parameters that will be requested from the end user as follows:

- **vip**—An IP address for the virtual server.
- We also use the **base_name** and **server_ips** parameters that were defined at [Step 2—Parameterize, page 24](#).

Parameters for a configuration template are defined using the **#param** directive. [Configuration Templates Directive Reference, page 85](#) describes directives including the **#param** directive.

Task: Add the **vip** parameter to the **create_vip.vm** template between the existing **base_name** and **server_ips** parameters.

Solution:

1. In the **Configuration Templates** table, click the [create_vip.vm](#) link to edit the template.
2. Add the following line:

```
#param($vip, 'type=ipv4', 'direction=in')
```



Note: Run the **create_vip.vm** template and verify that vDirect requests the **base_name**, **vip**, and **server_ips** parameters.

Step 3—Script

In this section we modify the remaining static configuration to use the parameters.

Task 1: Use the **\$base_name** variable to generate the virtual server identifier instead of the static names, and refer to the appropriate group identifier.

Solution:

1. We use the convention of **{base_name}_vip** for the virtual server identifier.
2. Modify the portion of the **create_vip.vm** template that you have just added as follows:

```
#set($virt_id = "${base_name}_vip")

/c/slb/virt $virt_id
    ena
    ipver v4
    vip 192.168.13.190
    vname "$base_name Vip"
/c/slb/virt ${virt_id}/service 80 http
    group $group_id
    rport 80
```

Task 2: Use the **\$vip** parameter instead of the static address.

Solution:

1. Replace the static IP address **192.168.13.190** with **\$vip**.
2. Modify the **create_vip.vm** template as follows:

```
#param($vip, 'type=ipv4', 'direction=in')
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

#set($group_id = "${base_name}_group")
#set($virt_id = "${base_name}_vip")

## Validation part
#set($found_group = $adc.readBean('enh/c/slb/group', $group_id))
#if($adc.isEmpty($found_group))
    #error("Group $group_id already exists!")
#end

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    #set($found_server = $adc.readBean('enh/c/slb/real', $server_id))
    #if($adc.isEmpty($found_server))
        #error("Server $server_id already exists!")
    #end
#end

## Configuration part
/c/slb/group $group_id
    ipver v4
    name "$base_name Group"

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    /c/slb/real $server_id
        ena
        ipver v4
        rip $server_ip
        name "$server_ip"

    /c/slb/group $group_id
        add $server_id
#end

/c/slb/virt $virt_id
    ena
    ipver v4
    vip $vip
    vname "$base_name Vip"

/c/slb/virt ${virt_id}/service 80 http
    group $group_id
    rport 80
```

Task 3: Fail if the virtual server existed before running the template.

Solution:

1. To read information from Alteon we use the methods of the **\$adc** variable. This is a special variable, called a Device Adapter, that represents the Alteon and can be used to query Alteon while the template is running. [Device Adapters, page 63](#) describes the **\$adc** functions.
2. To read configuration values from Alteon, specific beans (beans are simply a Java object with specially named methods) need to be used. [Device Adapters, page 63](#) describes how to identify bean classes. In addition to get the full list of available beans, please consult the vDirect Java docs.
3. Alteon version 29.0 introduced default alphanumeric identifiers for the server load balancing configuration. Since beans that map to the numeric configuration are different than beans that map to the alphanumeric configuration, add the prefix **/enh** to the standard CLI path.
4. To check if a virtual server exists, call **\$adc.readBean('/enh/c/slb/virt', \$virt_id)**.
5. To stop execution and return an error, use the **#error** directive.
6. Modify the **create_vip.vm** template as follows:

```
#param($vip, 'type=ipv4', 'direction=in')
#param($base_name, 'type=string', 'direction=in')
#param($server_ips, 'type=ipv4[]', 'direction=in')

#set($group_id = "${base_name}_group")
#set($virt_id = "${base_name}_vip")

## Validation part
#set($found_vip = $adc.readBean('enh/c/slb/virt', $virt_id))
#if($adc.isEmpty($found_vip))
    #error("Vip $virt_id already exists!")
#end

#set($found_group = $adc.readBean('enh/c/slb/group', $group_id))
#if($adc.isEmpty($found_group))
    #error("Group $group_id already exists!")
#end

#foreach($server_ip in $server_ips)
    #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
    #set($found_server = $adc.readBean('enh/c/slb/real', $server_id))
    #if($adc.isEmpty($found_server))
        #error("Server $server_id already exists!")
    #end
#end

## Configuration part
/c/slb/group $group_id
  ipver v4
  name "$base_name Group"
```



```
(continued)

#foreach($server_ip in $server_ips)
  #set($server_id = "${base_name}_${server_ip.replace('.', '_')}")
  /c/slb/real $server_id
    ena
    ipver v4
    rip $server_ip
    name "$server_ip"

  /c/slb/group $group_id
    add $server_id
#end

/c/slb/virt $virt_id
  ena
  ipver v4
  vip $vip
  vname "$base_name Vip"

/c/slb/virt ${virt_id}/service 80 http
  group $group_id
  rport 80
```

Example and Exercises—Deleting a Virtual Server

Goal: To delete the HTTP virtual server configuration that was created at [Example and Exercises—Creating a Virtual Server, page 27](#).

Develop Static Configuration

Create a working set of CLI commands that deletes the static Alteon configuration in the **Alteon_dump.txt** file.

The section of the configuration that handles the HTTP virtual server configuration is shown below:

```
/c/slb/real real-server-1
    ena
    ipver v4
    rip 192.168.12.10
    name "192.168.12.10"
/c/slb/real real-server-2
    ena
    ipver v4
    rip 192.168.12.11
    name "192.168.12.11"
/c/slb/real real-server-3
    ena
    ipver v4
    rip 192.168.12.12
    name "192.168.12.12"
/c/slb/group vDirect-servers-group
    ipver v4
    add real-server-1
    add real-server-2
    add real-server-3
    name "Group vDirect-servers-group"

/c/slb/virt vDirect-virtual-service
    ena
    ipver v4
    vip 192.168.13.190
    vname "vname vDirect-virtual-service"
/c/slb/virt vDirect-virtual-service/service 80 http
    group vDirect-servers-group
    rport 80
```

Parameterize & Script

This section describes how to turn the static text above into a configuration template.

Step 1—Creating a configuration template with static configuration

This section describes how to turn the static configuration above into a static set of commands that delete this configuration.

Task: Create a new **delete_vip.vm** template that deletes the configuration above.

Solution:

1. Log in to the vDirect Web user interface, as described at [Logging In to vDirect, page 18](#).
2. Create a new configuration template and name it **delete_vip.vm**, as described at [Creating Your First Configuration Template, page 19](#).
3. Paste the section of the configuration that handles the creation of the real servers and group into the **delete_vip.vm** text.
4. Modify the text to delete each and every configuration object including real servers, group, and virtual server.

```
/c/slb/virt vDirect-virtual-service
del
/c/slb/group vDirect-servers-group
del
/c/slb/real real-server-1
del
/c/slb/real real-server-2
del
/c/slb/real real-server-3
del
```

Step 2—Parameterize

In this section we turn the static text above into a configuration template. There are several possible approaches:

1. The template gets the identifiers of the virtual server, group, and real server. This may be efficient but is not very user friendly.
2. The template gets the same parameters as the **create_vip.vm** template, and calculates the identifiers from those parameters.
3. The template gets the base name, and discovers from Alteon the items that should be deleted.

Writing templates that implement approaches 1 and 2 is simpler but requires more information from the user. We will implement approach 3 and define **base_name** as the only parameter required.

Parameters for a configuration template are defined using the **#param** directive. [Configuration Templates Directive Reference, page 85](#) describes directives including the **#param** directive.

Task: Add the **base_name** parameter to the delete_vip.vm template.

Solution:

1. In the **Configuration Templates** table, click the [delete_vip.vm](#) link to edit the template.
2. Add the following as the first two lines:

```
#param($base_name, 'type=string', 'direction=in')
```



Note: Run the **delete_vip.vm** template and verify that vDirect requests the **base_name** parameter.

Step 3—Script

In this section we modify the remaining static configuration to use the parameters and query information from Alteon.

Task 1: Use the **\$base_name** variable to generate the virtual server identifier and group identifier, and to generate the deletion command.

Solution:

1. We use the convention of **{*\$base_name*}_vip** for the virtual server identifier.
2. We use the convention of **{*\$base_name*}_group** for the group identifier.
3. In the **delete_vip.vm** template, modify the section that deletes the virtual server and group as follows:

```
#param($base_name, 'type=string', 'direction=in')

/c/slb/virt ${base_name}_vip
    del
/c/slb/group ${base_name}_group
    del
/c/slb/real real-server-1
    del
/c/slb/real real-server-2
    del
/c/slb/real real-server-3
    del
```

Task 2: Read the real servers that are associated with the group and delete them.

Solution:

1. In the vDirect Java doc, locate the **SlbNewCfgEnhGroupRealServerEntry** bean. This bean maps the relationship between the group and real server.



Note: **RealServGroupIndex** and **ServIndex** hold the group identifier and the real server identifier respectively.

2. We read all the beans that have an index equal to the group identifier and then delete them.



Note: The **\$adc.readAll()** function returns all the beans that match the specified properties of the search bean provided. This allows us to query Alteon for all the real servers in the group.

3. Modify the **delete_vip.vm** template as follows:

```
#param($base_name, 'type=string', 'direction=in')

#set($group_id = "${base_name}_group")

#set($search_server = $adc.newBean('SlbNewCfgEnhGroupRealServerEntry'))
  ##calls the bean that maps a group to real servers

#set($search_server.RealServGroupIndex = $group_id)
  ##sets the group identifier

#set($found_servers = $adc.readAll($search_server))
  ##returns all beans that have an index matching the group identifier

#foreach($found_server in $found_servers)
  /c/slb/real $found_server.ServIndex
  del
  ##deletes real servers associated with the group

#end

/c/slb/virt ${base_name}_vip
  del
  ##deletes the virtual server

/c/slb/group $group_id
  del
  ##deletes the group
```

Task 3: Test the **delete_vip.vm** configuration template.

Solution:

1. Dry run the **delete_vip.vm** template with a non-existing **\$base_name** variable value.
The generated script should only show deletion of the virtual server and group because no real servers exist for the **\$base_name** variable.
2. Choose a base name and run the **create_vip.vm** template.
You can see the generated CLI transcript. You may also connect to Alteon and perform a configuration diff.
3. Dry run the **delete_vip.vm** template with the same **\$base_name** value used in [step 2](#).
The generated script should also show the deletion of the real servers.
4. Run the **delete_vip.vm** template with the same **\$base_name** value used in [step 2](#).
You can see the generated CLI transcript. You may also connect to Alteon and perform a configuration diff.
Verify that the diff shows no change. There should be no change since the **delete_vip.vm** template has deleted the configuration performed by the **create_vip.vm** template.

Troubleshooting Configuration Templates

This section describes two general methods for troubleshooting when your template does not behave as expected.

- Start commenting out lines of code using the following syntax:

```
## This is a comment
```

- Debug according to vDirect log level using the following syntax:

```
#log("<log level>", "<information to log>")
```

The following example illustrates these methods:

```
#foreach($server_ip in $server_ips)

    #set($server_id = $base_name + '_' + $server_ip.replace('.', '_'))

    ##
    ## The log message can be seen via vDirect UI
    ##
    #log("debug", "Server id is $server_id")

    #set($found_server = $adc.readBean('enh/c/slb/real', $server_id))

    #if($adc.isEmpty($found_server))
        #error("Server $server_id already exists!")
    #end

#end
```

For more information on using the #log directive, see [Log Directive, page 92](#).

CHAPTER 2 – WORKFLOW TEMPLATE BASICS

This section describes the following topics:

- [Introducing Workflow Templates, page 39](#)
- [Developing Workflow Templates, page 41](#)

Introducing Workflow Templates

Configuration templates are usually good for “one off” configuration tasks, where data persistence is not necessarily required. *Persistence* refers here to the data configured in devices and/or the data held within the configuration template itself, such as the IP addresses and names of the devices we wish to configure. The advantage of configuration templates is in their simplicity, which is based on the device CLI. However, a lack of persistence prevents the administrator from creating scripts that manage the entire life-cycle of a device.

To allow the administrator to manage more complex configuration scenarios, vDirect introduces the workflow template.

Workflow templates are a powerful tool that take ADC configuration to the next level by adding persistence. Workflow templates are higher-level scripts that utilize configuration templates as their building blocks. With workflow templates, a user can:

- Use outputs from one configuration template as inputs to the next configuration template in the chain.
- Save a workflow state (for example, the names of all configured devices and the parameters configured for them) in a persistent database.

A user can create a workflow template by simply combining a few configuration templates, passing the outputs of one template as inputs to the next. This enables users to create and maintain a complete ADC configuration and service life-cycle. Here are a few examples that highlight the need for workflow templates:



Examples

- A** We return to the example of validating the SSL certificates across a range of ADCs used in [Configuration Template Basics, page 15](#). The administrator now knows that the set of ADCs on which to validate SSL certificates on is not going to change for a while. In this case, the administrator does not want to retype the ADC names and IP addresses each time the script is run. With the use of workflow templates, the administrator needs to specify ADC IP addresses and names only once and they are preserved within the script for later runs. If the administrator adds, deletes, or modifies an ADC, the workflow template implements the changes on each subsequent run.
- B** Let's focus on the configuration of a “cross-device” service, one that requires the configuration of a few devices. An excellent example is the Radware service “DefenseSSL”. DefenseSSL is intended to provide DDoS protection on top of SSL encrypted traffic. In this scenario, Radware's DefensePro protects the organization but needs the help of Radware's Alteon to decrypt the SSL traffic. Configuration of this type of service requires specifying parameter values in DefensePro, then passing some of these values as inputs to Alteon configuration parameters, and vice versa. So the parameter values configured in one device must be “remembered” and passed on to a second device. In addition, the entire set of configured parameters, including the devices themselves (IP addresses and names) must also be preserved for later configuration changes. This represents an excellent use case for a workflow template.

Key Term Definitions

The following is a list of key terms used with vDirect workflows:

- **Workflow template**—A .zip archive file containing a set of actions of a single vDirect workflow. Workflow templates are installed in vDirect and can then be used by customers to create workflows of a given type. Customers may obtain workflow templates from Radware or they may author their own templates.
- **Workflow**—An instance of a workflow template. Each workflow is a unique instance of a workflow template that is stateful and persistent. Workflows are managed in vDirect using the vDirect Web user interface or programmatically through the vDirect SDK using customer written programs.
- **Action**—An executable process defined by the workflow template that transforms a workflow from one state to another state. Workflows in vDirect have a life cycle that begins when the workflow is created, and ends when the workflow is deleted. In between, the workflow template may support other state transitions that are activated by running actions on the workflow. These actions are not predefined by vDirect, but instead are completely customizable by the workflow template.

Benefits

The immediate benefits of powerful vDirect workflow templates are **efficient and error-free ADC life-cycle management**. vDirect and the workflow templates you run within it remember the state of each ADC and service you have created and configured. Updating or deleting these services and resources does not require the user to remember any configuration details. If you created a complex ADC service three months ago and now want to update or completely delete it, simply run the matching workflow template with a click of a button. The vDirect workflow template takes care of everything.

Figure 3: vDirect Workflow Template Example

```
<persist>
  <!-- Declare the persistent devices associated with the workflow -->
  <devices>
    <!-- This example just uses a single ADC that it configures -->
    <device name="adc"/>
  </devices>

  <!-- Declare the persistent parameters of the workflow -->
  <parameters>
    <!-- User Inputs -->
    <parameter name="vipAddress" type="ip" prompt="VIP address"/>
    <parameter name="serverIps" type="ip[]"/>
    <parameter name="complvl" type="int" min="1" max="9"/>

    <!-- Calculated -->
    <parameter name="virtId" type="int"/>
    <parameter name="compolId" type="string"/>
    <parameter name="groupId" type="int"/>
    <parameter name="serversByIp" type="object"/>
  </parameters>
</persist>
```


Developing Workflow Templates

In this section, you will learn the fundamentals of writing workflow templates.

This section describes the following topics:

- [Prerequisites, page 41](#)
- [Workflow Template Structure, page 41](#)
- [Workflow Template Development Steps, page 41](#)
- [Preparing Your Environment, page 42](#)
- [Managing Workflow Templates, page 42](#)
- [Troubleshooting Workflows, page 59](#)

Prerequisites

For a list of the prerequisites necessary for developing workflow templates are the same as for developing configuration templates. See [Prerequisites, page 17](#).

Workflow Template Structure

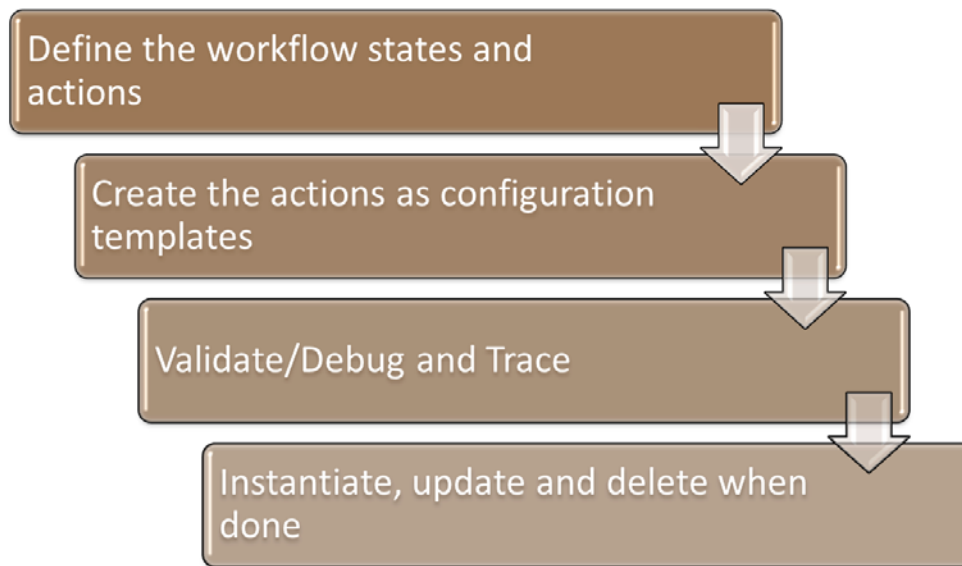
A workflow template is a collection of files, the most important of which is the **workflow.xml** *Workflow Descriptor* that defines the workflow (for example, by using persistent parameters, states, and actions), and other files that are used by the workflow template actions (such as configuration templates). A workflow template can be exported from and imported into vDirect in the form of a .zip archive that contains the workflow files.

The **workflow.xml** descriptor defines both the persistent data for the workflow, as well as the *actions* the user can run.

Workflow Template Development Steps

Radware recommends that you follow the steps shown in [Figure 4 - Recommended Workflow Template Development Steps, page 42](#) when developing a workflow template.

Figure 4: Recommended Workflow Template Development Steps



1. **Define the workflow states and actions**—Define which actions to use and in which cases they apply. In most cases, you will have an action that handles the initial creation of the configuration, and an action that handles the cleanup of the configuration. If you consider allowing updates to the configuration, there may be several additional actions that you can design.
2. **Create the actions as ADC configuration templates**—For each action, define and implement the appropriate steps. This usually involves executing configuration templates. At the end of this stage, you will have the configuration templates that you need, and you will have defined the parameters that need to be passed to the templates. The values that are passed to the configuration templates may also be remembered (persisted) in the workflow so that they can be used when calling during another action. For example, the action that deletes the configuration template does not accept user parameters and should have all the parameters it requires persisted.
3. **Validate/Debug and Trace**—You may need to check that your workflow template works as expected.
4. **Instantiate, Update and Delete Workflow Instances**—To use a workflow template, you create (instantiate) a workflow instance based on actions and states. You can then update the workflow instance by running actions, and eventually delete it where it is no longer needed.



Note: A *workflow instance* is often simply called a *workflow*. A *workflow* is thus the combination of a workflow template bound to a specific set of values of the persistent data defined by the template.

Preparing Your Environment

For information on how to set up vDirect and connect it to Alteon, see [Preparing Your Environment, page 18](#).

Managing Workflow Templates

This section introduces some aspects of the workflow template capabilities of the vDirect Web interface.

This section describes the following topics:

- [Viewing Workflow Templates, page 43](#)
- [Creating Your First Workflow Template, page 43](#)
- [Reviewing Your First Workflow Template, page 44](#)
- [Creating Your First Workflow Instance, page 44](#)
- [Updating the Action of Your First Workflow Instance From the Workflow Screen, page 45](#)
- [Deleting Your First Workflow Instance From the Workflow Screen, page 45](#)
- [Creating a Configuration Template for a Workflow Template Action, page 46](#)
- [Modifying an Action to Run a Configuration Template, page 46](#)
- [Creating a Stateful Workflow Template, page 54](#)
- [Upgrading XML Workflows, page 56](#)

Viewing Workflow Templates

This section describes how to view workflow templates that already exist in vDirect.



To view workflow templates

- > Select **Inventory > Workflow Templates**.

A table listing all the workflow templates defined in vDirect displays. If this is your first use, the table is empty.

Creating Your First Workflow Template

This section describes how to create your first workflow template.



To create your first workflow template

1. Select **Inventory > Workflow Templates**.
2. Click **New**.

The **New Workflow Template** dialog box displays initial **workflow.xml** content, as follows:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="REPLACE_WITH_NAME">

</workflow>
```

3. Click in the text area and modify the content as follows:
 - a. Change the value of the `name` attribute from `"REPLACE_WITH_NAME"` to `"idle"`.
 - b. Add an `<actions>` tag and close it.
 - c. Inside the `<actions>` tag, add the following:

```
<action name="init_idle"></action>
```

- d. The content of the **workflow.xml** file should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle">

  <actions>
    <action name="init_idle">
    </action>
  </actions>
</workflow>
```

4. Click **Save**.

If the **workflow.xml** syntax is valid, the **New Workflow Template** dialog box closes, and **idle** displays in the list of workflow templates with a **Valid** status indication.

Reviewing Your First Workflow Template

This section describes how to view the files accompanying your workflow template, and the actions and resulting state transitions for the workflow template.



To review your first workflow template

1. Select **Inventory > Workflow Templates**.
2. In the **Name** column, click the [idle](#) link.

The **Workflow Template** screen displays showing the following workflow information:

- The **Files** parameter lists the files contained in the workflow template.
The **workflow.xml** file is displayed.
 - The **State Transitions** table shows the available actions and resulting state transitions, as follows:
 - a. **createWorkflow**—A default empty action that does nothing. This action is called when a new workflow instance is created from the workflow template. The workflow instance state is “created” after the action is successfully complete.
 - b. **init_idle**—The action that we have defined. This action can be called when the workflow instance is “created”. The workflow instance remains “created” after the action is successfully complete.
 - The title, description (if any), tags (if any), status, and tenants of the workflow template.
3. Use the Back button of your browser to return to the **Workflow Templates** screen.

Creating Your First Workflow Instance

This section describes how to create your first workflow instance.



To create your first workflow instance

1. Select **Inventory > Workflow Templates**.
2. In the **Workflow Templates** table, select the **idle** workflow template, and click **Create Workflow**.
3. The **Create Workflow** screen displays.

4. In the **Name** field, type **idle_instance_01**.
5. Click **Run**.
The workflow status screen displays.
6. Click **Close**.
The **Workflow** screen for operating the workflow template displays the following information:
 - **Actions** table—Displays a history of actions run.
 - **Log Messages** table—Displays log entries that are automatically generated by vDirect in the context of the workflow instance, and log messages generated by the Workflow Template Developer either from log steps in the **workflow.xml** file, or from the **#log** directive in configuration templates activated during workflow template actions.
7. Select **Operations > Workflows** to access the new workflow.

Updating the Action of Your First Workflow Instance From the Workflow Screen

This section describes how to update a specific action (state) of a workflow instance.



To update an action of a workflow instance

1. Select **Operations > Workflows**.
2. In the table, select the box next to the **idle_instance_01** entry.
3. Click **Update**.
4. From the drop-down list of actions available for updating the workflow instance, select **init_idle**.
The **Update Workflow** screen displays.
5. Click **Run** to run the action.
The **Workflow** screen for operating the workflow template displays the following information:
 - The **Actions** table displays a new line representing the latest action run.
 - The **Log Messages** table displays the history of log messages for the workflow instance.
6. In the **Actions** table, select the box next to the **Idle_init** entry.
The **Log Messages** table displays only the logs generated by the selected action.

Deleting Your First Workflow Instance From the Workflow Screen

This section describes how to delete a workflow instance that is currently running. The procedure described here does not delete a workflow template.



To delete a workflow instance

1. Select **Operations > Workflows**.
2. In the table, select the box next to the **idle_instance_01** entry.
3. Click **Delete Workflow** and confirm the deletion.
When the operation is complete, the workflow instance will no longer display in the table.

Creating a Configuration Template for a Workflow Template Action

This section describes the general steps for creating a configuration template for use in a workflow template action.



To create a configuration template for a workflow template action

1. Create a new **idle2.vm** configuration template based on the **idle.vm** configuration template.
2. Read the idle time (using the **AgSystem** bean and its **AgCurCfgIdleCLITimeout** property) and return it via an output parameter (**prev_idle_time**).
3. Remove the apply and save commands from the **idle2.vm** configuration template.
4. Test **idle2.vm**.
5. Download **idle2.vm** for later use with your workflow template.

Your template should now look like this:

```
#param($idle_time, 'type=int', 'direction=in', 'prompt=Idle Time (Seconds)',  
'min=1', 'max=10080')  
#param($prev_idle_time, 'type=int', 'direction=out')  
  
#set($sys = $adc.readBean('AgSystem'))  
#set($prev_idle_time = $sys.AgCurCfgIdleCLITimeout)  
  
/c/sys/idle $idle_time
```



Notes

- The **prev_idle_time** parameter is an out parameter. This means that its value is returned when running the template.
- We use the **\$adc.readBean** function to read information back from the Alteon device.
- We use the **AgSystem** parameter to read a group of properties. We specifically need the **AgCurCfgIdleCLITimeout** property which is returned in the **prev_idle_time** output parameter.

Modifying an Action to Run a Configuration Template

This section describes how to modify the **init_idle** action to run the **idle2.vm** configuration template.

This section describes the following topics:

- [Adding Parameters to an Action, page 47](#)
- [Modifying an Action to Perform a Configuration Change, page 48](#)
- [Modifying a Workflow Template to Perform Create, Update and Delete Operations, page 50](#)

Adding Parameters to an Action

This section describes how to add parameters to the **init_idle** action.



To add parameters to the **init_idle** action

1. Select **Configuration > Workflow Templates**.
2. In the **Workflow Templates** table, click the [idle](#) link.
3. In the **Files** parameter, click the [workflow.xml](#) link.
4. Click in the text area and modify the content as follows:
 - In the `<inputs>` tag, add a `<devices>` tag and close it.
 - In the `<inputs>` tag, add a `<parameters>` tag and close it.
 - In the `<devices>` tag, add the following content:

```
<device name="adc" type="alteaon"/>
```

This specifies that Alteaon device needs to be provided as a parameter when calling the **init_idle** action.

- In the `<parameters>` tag, add the following content:

```
<parameter name="idle_time" type="int" prompt="Idle Time"/>
```

This specifies that an integer value specifying the required idle time needs to be provided when calling the **init_idle** action.

Your template should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle">

  <actions>
    <action name="init_idle">
      <inputs>
        <devices>
          <device name="adc" type="alteaon"/>
        </devices>
        <parameters>
          <parameter name="idle_time" type="int" prompt="Idle Time"/>
        </parameters>
      </inputs>
    </action>
  </actions>
</workflow>
```



Note: Remain in the **workflow.xml** text area. We continue to edit the **workflow.xml** file in the next section, [Modifying an Action to Perform a Configuration Change, page 48](#).

Modifying an Action to Perform a Configuration Change

This section describes how to modify the **init_idle** action to perform a configuration change.



To modify the **init_idle** action to perform a configuration change

1. Access the **workflow.xml** file as described in [Adding Parameters to an Action, page 47](#).
2. Click in the text area and modify the content as follows:

- After the `<inputs>` tag, add a `<sequence>` tag and close it.
This is where a sequence of steps will be defined. For a full list of available steps, see [Workflow Steps, page 121](#).
- In the `<sequence>` tag, add the following content to run the `idle2.vm` template:

```
<configuration file="idle2.vm"></configuration>
```

Your template should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle">

  <actions>
    <action name="init_idle">
      <inputs>
        <devices>
          <device name="adc" type="alteaon"/>
        </devices>
        <parameters>
          <parameter name="idle_time" type="int" prompt="Idle Time"/>
        </parameters>
      </inputs>
      <sequence>
        <configuration file="idle2.vm"></configuration>
      </sequence>
    </action>
  </actions>
</workflow>
```

3. Click **Save** in the bottom right corner of the screen.
4. Click the back arrow of your browser.

The **Workflow Template** screen displays.

5. The following validation error displays:

File 'idle2.vm' not found in step 'step' in action 'init_idle

6. Validate the **idle** workflow template as follows:
 - a. Select **Inventory > Configuration Templates**.
 - b. In the **Name** column, click the [idle2.vm](#) link.
 - c. In the **Configuration Template** screen, copy the content to your clipboard.
 - d. Select **Inventory > Workflow Templates** and verify that the status of the **idle** workflow template is **Invalid**.

- e. In the **Name** column, click the [idle](#) link.
 - f. In the **Workflow Template** screen, click **Create** and paste the content of the **idle2.vm** configuration template into the **idle** workflow template.
 - g. Click **Save** in the bottom right corner of the screen.
 - h. At the top of the screen, click the [HOME](#) link.
 - i. Select **Inventory > Workflow Templates** and verify that the status of the **idle** workflow template is **Valid**.
7. Create a workflow instance named **idle_instance02** from the **idle** workflow template as described at [Creating Your First Workflow Instance, page 44](#).
 8. Update the **idle_instance02** workflow instance and select the **init_idle** action.
vDirect prompts you to select an Alteon.
 9. Enter the idle time and run the workflow instance.
 10. Delete the **idle_instance02** workflow instance.
 11. Use SSH to connect to the Alteon and perform a **diff** operation.
 12. Verify that the change was not applied and saved.
 13. Revert the changes.
 14. Add a "commit" step that will handle the apply and save, as follows:
 - a. After the `</configuration>` tag, add a `<commit/>` tag. This step performs apply and save.
 - b. In addition, it is considered a good practice to add an error handler that will revert the changes.
 - c. After the `</ sequence>` tag, add an `<onError>` tag and close it. This defines a list of steps that vDirect performs when an error occurs anywhere in the action sequence.
 - d. In the `<onError>` tag, add an `<autoRevert/>` tag. This step performs revert or revert apply operations.
 - e. Save the **workflow.xml** file.
 15. You may want to create a new workflow instance and test that the update via the **init_idle** action performs the change which is also applied and saved.
Your template should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle">

  <actions>
    <action name="init_idle">
      <inputs>
        <devices>
          <device name="adc" type="alteaon"/>
        </devices>
        <parameters>
          <parameter name="idle_time" type="int" prompt="Idle Time"/>
        </parameters>
      </inputs>
      <sequence>
        <configuration file="idle2.vm"></configuration>
        <commit/>
      </sequence>
      <onError>
        <autoRevert/>
      </onError>
    </action>
  </actions>
</workflow>
```

Modifying a Workflow Template to Perform Create, Update and Delete Operations

This section describes how to modify the **idle** workflow template to perform a full create, update, and delete operation cycle.



To modify the idle workflow template to perform a full operation cycle

1. Copy and paste the **init_idle** action twice more and rename the actions **update_idle** and **restore_idle**.

We will add a `<persist>` section with persistent parameters and devices to remember the previous values, as well as to specify Alteaon only in **init_idle**.

2. Add the following section before `<actions>`:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle">

  <persist>
    <devices>
      <device name="adc" type="alteon"/>
    </devices>
    <parameters>
      <parameter name="start_idle_time" type="int"/>
      <parameter name="idle_time" type="int" prompt="Idle Time"/>
    </parameters>
  </persist>

  <actions>
```



Note: The **start_idle_time** parameter stores the idle time value read by the **init_idle** action. vDirect automatically maps the **idle_time** persisted parameter to the action input parameter and the template input parameter.

3. Modify the **init_idle** action to specify the Alteon and the **idle_time** without any additional attributes. The attributes are inherited from the persistent **idle_param** and **adc** definitions. Your template should now look like this:

```
<action name="init_idle">
  <inputs>
    <devices>
      <device name="adc"/>
    </devices>
    <parameters>
      <parameter name="idle_time"/>
    </parameters>
  </inputs>
```

4. Modify the **update_idle** action to require only **idle_time** as an input value. The **adc** value is provided from the persistent parameters.

It is not necessary to specify additional attributes since attributes are inherited from the persistent **idle_param** definition.

Your template should now look like this:

```
<action name="update_idle">
  <inputs>
    <parameters>
      <parameter name="idle_time"/>
    </parameters>
  </inputs>
```

5. Modify the **restore_idle** action to not require any input values, as follows. All parameter values are provided from the persistent parameters.

```
<action name="restore_idle">
  <sequence>
```

6. Modify the `<workflow>` tag to use **init_idle** as **createAction**, and **restore_idle** as **deleteAction**, as follows:

```
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle" createAction="init_idle" deleteAction="restore_idle">
```

7. Make **init_idle** and **restore_idle** invisible to the update operation so they can only be used when creating a new instance of the workflow and when deleting the workflow instance, as follows:

```
<action name="init_idle" visible="false">
<action name="restore_idle" visible="false">
```

8. We want the return value from the **idle2.vm** configuration template to be persisted into the **start_idle_time** parameter only after the **init_idle** action is complete. Since the out parameter from the template is called **prev_idle_time**, we use the **parameterMapping** capability available for configuration steps, as follows:

```
<sequence>
  <configuration file="idle2.vm">
    <parameterMapping>
      <map from="$start_idle_time" to="$prev_idle_time"/>
    </parameterMapping>
  </configuration>
</commit/>
```

9. When vDirect calls the **restore_idle** action, we want the **start_idle_time** parameter to pass to the **idle2.vm** configuration template. This means mapping **start_idle_time** to the **idle_time** parameter.

Add the following section to the **restore_idle** action:

```
<sequence>
  <configuration file="idle2.vm">
    <parameterMapping>
      <map from="$start_idle_time" to="$idle_time"/>
    </parameterMapping>
  </configuration>
</commit/>
```

Your template should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="idle" createAction="init_idle" deleteAction="restore_idle">

  <persist>
    <devices>
      <device name="adc" type="alteon"/>
    </devices>
    <parameters>
      <parameter name="start_idle_time" type="int"/>
      <parameter name="idle_time" type="int" prompt="Idle Time"/>
    </parameters>
  </persist>

  <actions>
    <action name="init_idle" visible="false">
      <inputs>
        <devices>
          <device name="adc"/>
        </devices>
        <parameters>
          <parameter name="idle_time"/>
        </parameters>
      </inputs>
      <sequence>
        <configuration file="idle2.vm">
          <parameterMapping>
            <map from="$start_idle_time" to="$prev_idle_time"/>
          </parameterMapping>
        </configuration>
        <commit/>
      </sequence>
      <onError>
        <autoRevert/>
      </onError>
    </action>
    <action name="update_idle">
      <inputs>
        <parameters>
          <parameter name="idle_time"/>
        </parameters>
      </inputs>
      <sequence>
        <configuration file="idle2.vm"> </configuration>
        <commit/>
      </sequence>
      <onError>
        <autoRevert/>
      </onError>
    </action>
  </actions>
</workflow>
```

(continued)

```

<action name="restore_idle" visible="false">
  <sequence>
    <configuration file="idle2.vm">
      <parameterMapping>
        <map from="$start_idle_time" to="$idle_time"/>
      </parameterMapping>
    </configuration>
    <commit/>
  </sequence>
  <onError>
    <autoRevert/>
  </onError>
</action>
</actions>
</workflow>

```

Creating a Stateful Workflow Template

In [Configuration Template Basics, page 15](#) we developed two configuration templates, **create_vip.vm** and **delete_vip.vm**, that can be used together to first create a configuration on Alteon and then later to undo or remove that configuration.

The **base_name** parameter provided by the user running the template is the key piece of information that links the two templates, and an identical value must be provided to both templates for them to work correctly as a pair. To run the **delete_vip.vm** configuration template separately at a later time, the administrator must remember the value assigned to the **base_name** parameter in the **create_vip.vm** configuration template and make sure that the same value is provided for the **delete_vip.vm** template.

Such a scenario is the simplest possible form of a *stateful configuration* where some parameters or changes associated with the configuration should be remembered to later modify, amend, or remove the configuration.

Instead of separately running the two configuration templates **create_vip.vm** and **delete_vip.vm**, we can create a workflow template named **manage_vip** that contains two actions named **create** and **delete**. The **delete** action automatically knows the value of the **base_name** parameter used by the **create** action so that the administrator does not have to provide this.

Exercise

Use the **create_vip.vm** and **delete_vip.vm** configuration templates to create a new workflow template named **manage_vip** that uses two states named **applied** and **removed**, and three actions named **create**, **apply**, and **remove**.

Make sure that when the workflow instance is created, the VIP configuration is applied on the Alteon, and that when the workflow instance is deleted, the VIP configuration is removed. Try to complete this exercise on your own and test that it works.

Your template should now look like this:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="manage_vip"
  createAction="create"
  deleteAction="remove">

  <description>
    A simple example workflow for managing a VIP. The parameters are provided
    when the workflow is created and then cannot be changed after. The
    configuration of the virtual server, group, and real servers can be applied
    or removed from the Alteon at anytime.
  </description>

  <persist>
    <!-- Declares the persistent devices associated with the workflow -->
    <devices>
      <device name="adc" type="alteon"/>
    </devices>

    <!-- Declares the persistent parameters of the workflow -->
    <parameters>
      <parameter name="base_name" type="string" prompt="Base name"/>
      <parameter name="vip" type="ipv4" prompt="VIP address"/>
      <parameter name="server_ips" type="ipv4[]" minLength="1"
        prompt="Real server IP addresses"/>
    </parameters>
  </persist>

  <!-- Declare the states used by this workflow. The workflow actions cannot
    reference a state unless it is defined here -->
  <states>
    <state name="applied">
      <description>
        The VIP configuration is applied on the alteon
      </description>
    </state>
    <state name="removed">
      <description>
        The VIP configuration is removed from the alteon
      </description>
    </state>
  </states>
```

```
(continued)

<!-- Declare the workflow actions -->
<actions>

  <action name="create" toState="applied" visible="false">
    <inputs>
      <devices>
        <device name="adc"/>
      </devices>
      <parameters>
        <parameter name="base_name"/>
        <parameter name="vip"/>
        <parameter name="server_ips"/>
      </parameters>
    </inputs>
    <sequence >
      <configuration name="create" file="create_vip.vm"/>
      <commit/>
    </sequence>
  </action>

  <action name="apply" fromState="removed" toState="applied">
    <sequence>
      <configuration name="create" file="create_vip.vm"/>
      <commit/>
    </sequence>
  </action>

  <action name="remove" fromState="applied" toState="removed">
    <!-- Note that no inputs are needed to run the remove action. Also note
    that when the workflow is deleted the remove action will only execute if
    the current state is "applied" -->
    <sequence>
      <configuration name="delete" file="delete_vip.vm"/>
      <commit/>
    </sequence>
  </action>
</actions>
</workflow>
```

Upgrading XML Workflows

When making changes to existing workflow templates, be careful to avoid changes that are incompatible with existing instances of the old workflow. Workflow template versions in the **workflow.xml** file are used to indicate when the version changes.



Note: vDirect does not allow you to install an older version of a workflow template over a newer version.

If the developer makes changes that cause the persistent data stored for existing workflow instances to be incompatible with the new workflow template, the developer can include a Groovy script as part of the workflow to directly modify the old persisted data to make compatible with the new workflow. vDirect invokes this script after the new template version is installed either when the upgrade action is explicitly requested, or automatically when a different action is requested and the workflow instance has not yet been upgraded. Workflow instances are upgraded “on demand” rather than all at once when the new workflow template is installed.



Note: It is entirely the responsibility of the workflow developer to ensure that data is migrated properly. The script is invoked with several properties in the script binding.

Table 1: Workflow Upgrade
ObjectsM:\vDirect\templates_and_workflows\working\cfgWorkflows.fm

Object	Description
workflowVersion	The version of the existing workflow that is being upgraded. The script might use this to make decisions about how to upgrade the workflow.
workflowState	Contains the state and persistent parameters of the existing workflow that is being upgraded. The workflowState object has the following properties: <ul style="list-style-type: none">• name—The name of the workflow instance being upgraded.• state—The current state of the existing workflow instance.• parameters—A dictionary of the existing persistent parameters.
vdirect	A VDirectServerClient object that can be used if needed to invoke vDirect APIs.



Note: The script must return a **WorkflowState** object that is to be used as the new state. It can be the provided **!workflowState** object that is modified, or it can be a new object. The script can change both the persistent parameters and the state if needed.



Example

The following workflow template is in use, and there is an existing workflow from this template:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="test-xml-upgrade"
  version="1.0.0">

  <persist>
    <parameters>
      <parameter name="a" type="int" min="20"/>
      <parameter name="b" type="int"/>
      <parameter name="c" type="int"/>
    </parameters>
  </persist>
</workflow>
```

(continued)

```
<actions>
  <action name="test">
    <inputs>
      <parameters>
        <parameter name="a"/>
        <parameter name="b"/>
        <parameter name="c"/>
      </parameters>
    </inputs>
  </action>
</actions>

</workflow>
```

The developer creates a new version of this workflow template that is incompatible with the existing workflow instance:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="test-xml-upgrade"
  version="2.0.0"
  upgradeScript="upgrade.groovy">

  <persist>
    <parameters>
      <parameter name="a" type="int" min="40"/>
      <parameter name="b" type="int"/>
      <parameter name="d" type="int" defaultValue="30"/>
    </parameters>
  </persist>

  <actions>
    <action name="test">
      <inputs>
        <parameters>
          <parameter name="a"/>
          <parameter name="b"/>
          <parameter name="d"/>
        </parameters>
      </inputs>
    </action>
  </actions>

</workflow>
```

The following changes have occurred in the new template:

- The minimum allowed value for persistent parameter **a** has changed from 20 to 40.
- Parameter **c** is no longer used.
- A new parameter **d** has been added with a default value of 30.

The problem for the upgrade is that existing workflow instances have no value for **d**, and their value for **a** may be less than the allowed minimum of 40.

The upgrade script might be as follows:

```
// Modify parameter a for the new minimum value
if (workflowState.parameters['a'] < 40)
    workflowState.parameters['a'] = 40

// Add new parameter d
workflowState.parameters['d'] = 30

// Remove unused parameter c
workflowState.parameters.remove('c')

// Return the existing parameters with modifications
workflowState
```



Note: The new workflow template uses the **upgradeScript** attribute to specify the name of the upgrade script to use. In this case the **upgrade.groovy** file should be packaged with the new template.

Troubleshooting Workflows

This section describes two general methods for troubleshooting a workflow.

The following facilities are available:

- Using the `#log` directive in a configuration template, as described at [Troubleshooting Configuration Templates, page 37](#).

For more information on using the `#log` directive, see [Log Directive, page 92](#).

- Using the `<log>` step in a workflow template sequence.
- After running a configuration step there are two variables available:

```
__generatedScript
__cliTranscript
```

You can log these values or save them in a persistent property, as follows:

```
<sequence>
  <configuration name="configure" file="configure.vm"/>
  <set saveAs="$script" value="$__generatedScript"/>
  <log "$__cliTranscript"/>
</sequence>
```


CHAPTER 3 – COMMON VELOCITY TEMPLATE LANGUAGE (VTL) SYNTAX

Velocity is a library designed for generating Web pages through the use of templates. Radware has adapted the concept of template text files to be used in dynamically generating CLI commands. This section describes some commonly used features of the Velocity Template Language (VTL).

For a full description of the VTL syntax, see <http://velocity.apache.org/engine/devel/user-guide.html> and <http://velocity.apache.org/engine/devel/vtl-reference-guide.html>.

This section describes the following topics:

- [Variables, page 61](#)
- [Properties, page 61](#)
- [Setting Variables, page 62](#)
- [Using Logical Expressions, page 62](#)
- [Iterating Through a List, page 62](#)

Variables

Variables are used to contain values. When a template executes, all of the input parameters exist as variables, and the template can freely reference these variables as well as create new variables.

Variable names must begin with a letter and may include alphanumeric characters as well as “-” and “_”. To reference a variable anywhere in the template, use the **\$** reference operator followed by the name of the variable. In some cases the reference may be ambiguous unless you use the full formal reference notation of putting braces around the variable name **\${}**. If the variable might not exist or may be null, you can also use the silent operator **!** in front of the variable name.



Examples

- A** `$adcService`
- B** `$!adcService`
- C** `${adcService}`
- D** `$!{adcService}`

Properties

Properties are values that can be obtained out of an object.



Examples

- A** `$customer.Address`
- B** `${purchase.Total}`

Setting Variables

You set variables with the `#set` directive.



Example

```
#set($var1 = $foo + 1)
```

Using Logical Expressions

You use logical expressions with the `#if`, `#elseif`, `#else`, and `#end` directives.



Example

```
#if($foo == $bar)
it's true!
#else
it's not!
#end
```

Iterating Through a List

You iterate through a list with the `#foreach` directive in the following syntax:

```
#foreach ($ref in arg)  #end
```



Examples

A

```
#foreach ($item in $items)
... statements ...
#end
```

B

```
#foreach ($item in ["Not", $my, "fault"])
... statements ...
#end
```

C

```
#foreach ($item in [1...3])
... statements ...
#end
```

CHAPTER 4 – DEVICE ADAPTERS

For each `#device` directive in the template, vDirect creates a *Device Adapter* and stores this as a variable in the Velocity context. The template can use this device variable to communicate with the device using the vDirect Device SDK during template execution. The name of this variable is the same as the name argument provided to the `#device` directive.

When the template has no `#device` directive, one is implicitly added for an Alteon device as follows: `#device($adc, 'alteon')`.

This means that by default there is a single device variable named `$adc`.



Note: Radware recommends that you always include the **#device** directive in your templates.

This section describes the following topics:

- [Device Variables, page 63](#)
- [Identifying Bean Classes, page 66](#)
- [Using Multiple Devices, page 68](#)
- [CLI-based Error Detection, page 70](#)

Device Variables

The behavior of the device variable may vary depending on the type of device being configured. The following device adapter methods can be used by the template:

Table 2: Available Device Adapter Methods

Method	Description
Object newBean(String beanClassOrCLIPath)	Creates an empty instance of one of the vDirect Device SDK JavaBeans supported by the device. The name supplied can be either the simple class name of the bean to create, or it can be a CLI command path if supported by the device. For more information, see Identifying Bean Classes, page 66 . The object returned is an instance of the vDirect bean, and the bean methods can then be called for it.
Object readBean(String beanClassOrCLIPath)	Same as Object newBean, but it reads the bean from the device and the returned object with all the values defined.
Object read(Object bean)	Reads the bean object specified from the device and returns an object of the same type with all the values defined.
List<?> readAllBeans(String beanClassOrCLIPath)	Reads all the entries in a table from the device and returns them as a list.
Object findFirst(Object bean)	Finds the first object on the device that matches the filled in fields of the bean specified.
List<?> findAll(Object bean)	Finds all the objects on the device that match the filled in fields of the bean specified.

Table 2: Available Device Adapter Methods

Method	Description
Object getFirstFreeIndex(String beanClassOrCLIPath)	Finds the first free index in the specified table.
Object getFreeIndex(Object bean)	Finds the first free index in a table after the bean specified.
Object getFreeIndex(String beanClassOrCLIPath, Object startValue)	Finds the first free index in the specified table after the start value.
Object getFreeIndexWithDefault(Object bean)	Same as getFreeIndex, but it returns the bean specified if its index is free.
Object getFreeIndexWithDefault(String beanClassOrCLIPath, Object startValue)	Same as getFreeIndex, but it uses the specified start value if it is available in the table.
boolean exists(Object bean)	Returns true if the specified table entry exists on the device; false otherwise.
boolean isEmpty(Object bean)	Returns true if the specified bean is null or empty.
boolean isEmpty(Object bean)	Returns true if the specified bean is not null and not empty
boolean isNull(Object bean)	Convenience method to test if a returned bean object is null.
boolean isNotNull(Object bean)	Convenience method to test if a returned bean object is not null.
boolean setEnumValue(Object bean, String property, String value)	Sets the value of a JavaBean property when the property is a Java enum type. This method allows the template to simply use the string name of the desired enum value.
Iterator iterate(String beanClassOrPath)	Returns a table iterator that can be used with the Velocity <code>#foreach</code> directive.
String getReply()	Holds the device response to the last CLI command sent.
boolean isAlteonMaster()	Returns true if the device is an Alteon, is reachable, and its HA state is NONE, MASTER, or ACTIVE.
boolean isNetworkReachable(int timeout)	Returns true if the device is network reachable within the specified timeout in milliseconds. Reachable means ICMP ECHO or some equivalent test.
boolean testConfigProtocol(timeout)	Returns true if the configuration protocol specified for the device is working. This means the device is responding and the credentials vDirect is using are correct. The protocol is either HTTP or SNMP depending on the registered device configuration. The timeout is in milliseconds.
String getDeviceName()	Returns a name of the registered device from the vDirect inventory. If there is no backing registered device, the host name is returned.

As a rule, template authors should use the available CLI commands on the device to make configuration changes, and use the device variable to read values from the device in a way that does not require parsing of CLI output. However, in certain rare cases it may be easier to use the vDirect Device SDK rather than CLI commands to make configuration changes. For this reason, the following methods are also available:

- `void update (Object bean)`—Updates the object on the device. The object passed must be one of the vDirect factory beans.
- `void delete (Object bean)`—Removes a table entry from the device. The object passed must be one of the vDirect beans representing a table entry.
- `void apply()`—Applies the configuration on the device. The meaning of this method depends on the actual device. For Alteon, the new configuration becomes the current configuration. For DefensePro, the policies are updated.
- `void commit ()`—Commits the configuration on the device. The meaning of this method depends on the device. For Alteon, the new configuration is applied and then saved. For DefensePro, the policies are updated.
- `void create (Object bean)`—Creates the specified table entry on the device.

DefensePro Variables

The following Device Adapter methods can be used when the device is a DefensePro.

Table 3: Available DefensePro Adapter Methods

Method	Description
<code>ExportServerPolicyOptions getExportServerPolicyOptions ()</code>	Use to configure options for exporting DefensePro server policies.
<code>ImportExportResult exportServerPolicy (String policyName)</code>	Export the named policy from the DefensePro.
<code>ExportNetworkPolicyOptions getExportNetworkPolicyOptions ()</code>	Use to configure options for exporting DefensePro network policies.
<code>ImportExportResult exportNetworkPolicy (String policyName)</code>	Export the named policy from the DefensePro.
<code>ExportSubdomainsWhitelistOptions getExportSubdomainsWhitelistOptions()</code>	Use to configure options for exporting DefensePro sub-domain whitelists.
<code>ImportExportResult exportSubdomainsWhitelist (String policyName, String dnsProfileName)</code>	Export the whitelist for the named policy from the DefensePro.
<code>ImportPolicyOptions getImportPolicyOptions ()</code>	Use to configure common import options for server and network policies.
<code>ImportExportResult importNetworkPolicy (String policy)</code>	Import the previously exported policy to the DefensePro.
<code>ImportExportResult importServerPolicy (String policy)</code>	Import the previously exported policy to the DefensePro.
<code>ImportSubdomainsWhitelistOptions getImportSubdomainsWhitelistOptions()</code>	Use to configure options for importing a sub-domains whitelist.
<code>ImportExportResult importSubdomainsWhitelist (String policyName, String dnsProfileName, String whitelist)</code>	Import a previously exported whitelist for the specified policy and DNS profile.

Identifying Bean Classes

The device adapter methods that require the name of a factory bean class support two options for specifying the bean class. The first method is to use the Java class name of the corresponding vDirect factory bean without any package prefix. The second method is to use an equivalent Alteon CLI menu path that represents the same bean. vDirect provides a built-in mapping between some of the Java bean classes and their corresponding Alteon CLI menu paths. However, not all beans in vDirect have a mapping.

The following table shows the vDirect bean classes that have corresponding Alteon CLI menu paths:

Table 4: vDirect Bean Classes and Corresponding Alteon CLI Menu Paths

Java Factory Bean Class	Corresponding CLI Menu Path
AgSystem	/boot/reset
AgTftp	/cfg/gtcfg
LacpNewPortCfgTableEntry	/cfg/l2/lacp/port
StgNewCfgTableEntry	/cfg/l2/stg
PortTeamNewCfgTableEntry	/cfg/l2/team
TrunkGroupNewCfgTableEntry	/cfg/l2/trunk
VlanNewCfgTableEntry	/cfg/l2/vlan
IpNewCfgStaticArpEntry	/cfg/l3/arp/static
BgpGeneral	/cfg/l3/bgp
BgpNewCfgAggrEntry	/cfg/l3/bgp/aggr
BgpNewCfgPeerEntry	/cfg/l3/bgp/peer
DnsCfg	/cfg/l3/dns
IpNewCfgGwEntry	/cfg/l3/gw
IpNewCfgIntfEntry	/cfg/l3/if
OspfGeneral	/cfg/l3/ospf
OspfNewCfgAreaEntry	/cfg/l3/ospf/aindex
OspfNewCfgIntfEntry	/cfg/l3/ospf/if
RipNewCfgIntfEntry	/cfg/l3/rip
IpNewCfgStaticRouteEntry	/cfg/l3/route
VrrpNewCfgIfTableEntry	/cfg/l3/vrrp/if
VrrpGeneral	/cfg/l3/vrrp/track
VrrpNewCfgVirtRtrTableEntry	/cfg/l3/vrrp/vr
VrrpNewCfgVirtRtrVrGrpTableEntry	/cfg/l3/vrrp/vrgroup
MirrPortMirr	/cfg/pmirr
PmNewCfgPortMonitorEntry	/cfg/pmirr/monport
AgPortNewCfgTableEntry	/cfg/port
SlbNewAcclCfgCachePolEntry	/cfg/slb/accel/caching/cachepol
SlbNewAcclCfgCacheUrlListEntry	/cfg/slb/accel/caching/urllist
SlbNewAcclCfgCacheUrlRuleEntry	/cfg/slb/accel/caching/urllist/rule
SlbNewAcclCfgCompBrwsListEntry	/cfg/slb/accel/compress/brwslist
SlbNewAcclCfgCompBrwsRuleEntry	/cfg/slb/accel/compress/brwslist/rule

Table 4: vDirect Bean Classes and Corresponding Alteon CLI Menu Paths

Java Factory Bean Class	Corresponding CLI Menu Path
SlbNewAcclCfgCompPolEntry	/cfg/slb/accel/compress/compol
SlbNewAcclCfgCompUrlListEntry	/cfg/slb/accel/compress/urlist
SlbNewAcclCfgCompUrlRuleEntry	/cfg/slb/accel/compress/urlist/rule
HcsNewCfgTableEntry	/cfg/slb/advhc/script
SnmpHcNewCfgTableEntry	/cfg/slb/advhc/snmpHc
FltNewCfgTableEntry	/cfg/slb/flt/adv
SlbNewCfgGroupEntry	/cfg/slb/group
GslbGeneralCfg	/cfg/slb/gslb
GslbNewCfgRuleTableEntry	/cfg/slb/gslb/rule
GslbNewCfgRemSiteTableEntry	/cfg/slb/site
Layer7NewCfgHttpModListEntry	/cfg/slb/layer7/httpmod
SlbUrlBalance	/cfg/slb/layer7/loadbalance
SlbNewCfgUrlHttpMethodsTableEntry	/cfg/slb/layer7/slb/deletemeth
SlbUrlRedir	/cfg/slb/layer7/urlredir
SlbNewCfgDRecordEntry	/cfg/slb/linklb/drecord/entry
PipNewCfgTableEntry	/cfg/slb/pip
SlbNewCfgPortEntry	/cfg/slb/port
SlbNewCfgRealServerEntry	/cfg/slb/real
SlbNewSslCfgAuthPolEntry	/cfg/slb/ssl/authpol
SlbNewSslCfgAuthPolEntry	/cfg/slb/ssl/authpol/passinfo
SlbNewSslCfgAuthPolEntry	/cfg/slb/ssl/authpol/validity
SlbSslCfgCertsDefaults	/cfg/slb/ssl/certs
SlbNewSslCfgGroupsEntry	/cfg/slb/ssl/certs/group
SlbNewSslCfgSSLPolEntry	/cfg/slb/ssl/sslpol
SlbNewSslCfgSSLPolEntry	/cfg/slb/ssl/sslpol/passinfo
SyncGeneralCfg	/cfg/slb/sync
SlbNewCfgPeerEntry	/cfg/slb/sync/peer
SlbNewCfgVirtualServerEntry	/cfg/slb/virt
SlbNewCfgVirtServicesEntry	/cfg/slb/virt/service
AgRadiusConfig	/cfg/sys/radius
AgTacacsConfig	/cfg/sys/tacacs
VADCNewCfgTableEntry	/cfg/vadc
LacpInfoPortTableEntry	/info/l2/lacp
StpInfoTableEntry	/info/l2/stg
VlanInfoTableEntry	/info/l2/vlan
IntfInfoEntry	/info/l3/ip
OspfIntfInfoEntry	/info/l3/ospf/if
Rip2GeneralInfo	/info/l3/rip

Table 4: vDirect Bean Classes and Corresponding Alteon CLI Menu Paths

Java Factory Bean Class	Corresponding CLI Menu Path
PortInfoTableEntry	/info/link
SlbPortInfoEntry	/info/slb/port
SlbRealServerInfoEntry	/info/slb/real
SlbVirtServicesInfoEntry	/info/slb/virt
SwKeyInfo	/info/swkey
Layer4Oper	/oper/slb/clear
SlbOperGroupRealServerEntry	/oper/slb/group
SlbOperRealServerEntry	/oper/slb/real
ArpStats	/stats/l3/arp
IfStatsEntry	/stats/l3/ifClear
Ip6Stats	/stats/l3/ip6
ClearStats	/stats/l3/ipClear
RouteStats	/stats/l3/route
VrrpStats	/stats/l3/vrrp
MpCpuStats	/stats/mp/cpu
HttpStats	/stats/slb/http
SpStatsCpuUtilTableEntry	/stats/sp/cpu
MgmtStats	/stats/sys/mgmt
NtpStats	/stats/sys/ntp

Using Multiple Devices

By default, templates run against a single Alteon device. All of CLI commands in the template are sent to this device and the context variable works as the device adapter. However, it is possible to work with two or more different devices at the same time within a single configuration template.

In this case there are multiple device adapters, and the template must select which device its CLI commands should be directed to using a `#select` directive. Examples of a template communicating with more than one device include migrating configuration from one device to another, or configuring both the VX global administrator and a vADC at the same time.

To enable multiple device support, the template must include at least two `#device` directives that define each device connection required to run the template.

- For more information about the `#select` directive, see [Select Directive, page 98](#).
- For more information about the `#device` directive, see [Device Directive, page 87](#).

The following example shows how a single configuration template can be used to enable peer synchronization between two Alteon devices, named `peer1` and `peer2` in the template. It also demonstrates the `#log` and `#error` directives.



Example

```
##
## Sync configuration between two ADC devices
##

#property('description', 'Synchronize configuration between two ADCs')
#device($peer1, "prompt=First peer device")
#device($peer2, "prompt=Second peer device")

#param($syncSessionState, 'bool', 'in', 'prompt=Synchronize persistent session
state')
#param($updatePeriod, 'int', 'in', 'min=1', 'max=60', 'prompt=Update period')

#macro(sync, $adc, $peerIp, $syncSessionState, $updatePeriod)

    ## Convert the boolean value to the flag to use in the CLI
    #if($syncSessionState)
    #set($sessionStateFlag = 'enable')
    #else
    #set($sessionStateFlag = 'disable')
    #end

    ## Check to make sure this peer is not already defined on the device
    #set($search = $adc.newBean('/c/slb/sync/peer'))
    #set($search.ipAddr = $peerIp)
    #set($entry = $adc.findFirst($search))
    #if($adc.isNotNull($entry))
        ## Reuse the same entry
        #log('warning', "Peer $peerIp already exists on device $adc.ip")
        #set($freeIndex = $entry.index)
    #else
        ## Use the first available free index
        #set($entry = $adc.getFreeIndex($search))
        #set($freeIndex = $entry.index)
    #end

    ## ----- Configure configuration synchronization for the peer -----
    /c/slb/sync
        filt enable

    ## we disable the ports processing as they should be set by the initial
configuration and might not be similar between the two boxes
    ## this could be a parameter d/e
        ports disable

    ## we disable priorities as we believe that the different boxes should
have specific different priorities that governs who is the master and who is
the backup
        prios disable
```

```
(continued)

    ## proxy IPs configurations can only be supported if working in Hot
    Standby which we are not supporting at this stage
        pips disable
    ## we are not sure why this is needed and the default is disabled
        peerpips disable
        bwm enable
    ## state and update period should be set by application setting that
    requires persistent sessions synchronization ex: SSL ID, Client IP and cookie
    (pbind)
        state $sessionStateFlag
        update $updatePeriod
    ## the certificates should be loaded by the initial configuration and not
    synchronized in plain text over the sync protocol
        certs disable
    ## since certificate synchronization is disabled, no need for a pass
    phrase
        ##passphrs
        /c/slb/sync/peer $freeIndex
        #if ($adc.reply.toLowerCase().contains('error'))
            ## This means all the peer slots are in use!
            #error("No available peer slots on device $adc.ip")
        #end
        ena
        addr $peerIp
#end

## Configure the first peer
#select($peer1)
#sync($peer1, $peer2.ip, $syncSessionState, $updatePeriod)

## Configure the second peer
#select($peer2)
#sync($peer2, $peer1.ip, $syncSessionState, $updatePeriod)
```

CLI-based Error Detection

`DeviceAdaptor` variables have two methods that template developers can use at any time to test if the previous command resulted in an error response:

- `boolean isErrorResponse ()` —This property can be used after any command to verify if the last response was a device error, provided that device replies are being captured.
- `boolean isErrorResponse (String deviceResponse)` —Use this version with the `expect` facility to verify if the specified device response indicates an error when the template is fully in control of processing device replies.
- `#haltOnDeviceError(true|false) ... #end`—This block directive can be used to surround a block of commands. Every command is automatically tested for errors and, if an error response is detected, the template is halted with an exception.

- `#setHaltOnDeviceError(true|false)` — Turns automatic response checking on or off. Can be used at the top of a template to enable the feature for the entire template.



Note: Enabling automatic checking requires vDirect to examine every response from the device for errors and can cause the template to run significantly slower.

Automatic checking is disabled by default and must be enabled using one of the above directives.

CHAPTER 5 – STANDARD AND RADWARE VELOCITY TOOLS

Tools are predefined variables in the template context that perform utility tasks. vDirect populates the context with tools provided by the Velocity Project as well as custom Radware tools.



Note: If the template sets a context variable with the same name as a predefined tool, it hides the tool in the context and makes it unavailable for use.

This section describes the following topics:

- [Velocity Tools in Configuration Templates, page 73](#)
- [Radware Velocity Tools in Configuration Templates, page 73](#)

Velocity Tools in Configuration Templates

[Table 5 - Standard Velocity Tools, page 73](#) lists the Velocity Project tools that are available for use in configuration templates. Documentation for these tools is available at <http://velocity.apache.org/tools/devel/javadoc/org/apache/velocity/tools/generic/>.

Table 5: Standard Velocity Tools

Tool	Variable Name
Class Tool	\$class
Date Tool	\$date
Display Tool	\$display
Escape Tool	\$esc
Field Tool	\$field
Math Tool	\$math
Number Tool	\$number
Render Tool	\$render
Sorter Tool	\$sorter

Radware Velocity Tools in Configuration Templates

[Table 5 - Standard Velocity Tools, page 73](#) lists the Radware tools that vDirect provides.

Table 6: Radware Tools

Tool	Variable Name	Link
Converter Tool	\$convert	Converter Tool, page 74
Digest Tool	\$hash	Digest Tool, page 75
IP Address Tool	\$ipAddress	IP Address Tool, page 77
PEM Tool	\$pem	PEM Tool, page 79

Table 6: Radware Tools

Tool	Variable Name	Link
Subnet Tool	\$subnet	Subnet Tool, page 82

Converter Tool

The Converter Tool supports conversion of strings to various formats as the following example shows:

```
## Example $convert tool

#mockdevice(true)
#param($content, 'string', 'in', 'prompt=String to comput digest for')

#set($digest = $hash.getDigest('SHA-256', $content))
#set($base64 = $convert.printBase64($digest))
#set($hex = $convert.printHex($digest))

BASE64:
    $base64

HEX:
    $hex

#set($bytes1 = $convert.parseHex($hex))
#set($bytes2 = $convert.parseBase64($base64))

BASE64 BYTES:
    $display.list($bytes2)

HEX BYTES:
    $display.list($bytes1)
```

The Converter Tool has the following methods:

```
/**
 * Converts the string argument into an array of bytes.
 * @param base64Binary
 *     A string containing lexical representation
 *     of xsd:base64Binary.
 * @return
 *     An array of bytes represented by the string argument.
 * @throws IllegalArgumentException if string parameter does not conform to
 * lexical value space defined in XML Schema Part 2: Datatypes for
 * xsd:base64Binary
 */
```

```

(continued)

public byte[] parseBase64 (String base64Binary)

/**
 * Converts the string argument into an array of bytes.
 * @param hexBinary
 *     A string containing lexical representation of
 *     xsd:hexBinary.
 * @return
 *     An array of bytes represented by the string argument.
 * @throws IllegalArgumentException if string parameter does not conform to
 * lexical value space defined in XML Schema Part 2: Datatypes for xsd:hexBinary.
 */
public byte[] parseHex (String hexBinary)

/**
 * Converts an array of bytes into a string.
 * @param val
 *     An array of bytes
 * @return
 *     A string containing a lexical representation of xsd:base64Binary
 * @throws IllegalArgumentException if <tt>val</tt> is null.
 */
public String printBase64 (byte[] val)

/**
 * Converts an array of bytes into a string.
 * @param val
 *     An array of bytes
 * @return
 *     A string containing a lexical representation of xsd:hexBinary
 * @throws IllegalArgumentException if <tt>val</tt> is null.
 */
public String printHex ( byte[] val )

```

Digest Tool

The Digest Tool allows the template to compute the MD5, SHA-1, and SHA-256 hashes as the following example shows.

The hashes available depend on the installed JRE version.

```

## Example SHA-256 Digest
#mockdevice(true)
#param($content, 'string', 'in', 'prompt=String to comput digest for')
#param($digest, 'string', 'out')

#set($digest = $hash.getStringDigest('SHA-256', $content))

```

The Digest Tool has the following methods:

```

/**
 * Get a list of supported digest algorithms
 * @return A list of supported digest algorithms
 */
public List<String> getAlgorithms ();

/**
 * For advanced use not covered by this tool, get an instance of
 * MessageDigest
 * @param algorithm The digest algorithm to use
 * @return A MessageDigest
 * @throws NoSuchAlgorithmException If the algorithm is not supported
 */
public MessageDigest getDigestor (String algorithm) throws
NoSuchAlgorithmException;

/**
 * Calculate a digest.
 * @param algorithm The algorithm to use, e.g. SHA-256
 * @param content The message to digest. This can be a byte[] or a String. If
it is a String, the UTF-8 bytes will be used.
 * @return The message digest as a byte[]
 * @throws UnsupportedEncodingException If UTF-8 is not supported (unlikely)
 * @throws NoSuchAlgorithmException If the requested algorithm is not supported
 */
public byte[] getDigest (String algorithm, Object content) throws
UnsupportedEncodingException, NoSuchAlgorithmException;

/**
 * Calculate a digest and return it as a HEX string.
 * @param algorithm The algorithm to use, e.g. SHA-256
 * @param content The message to digest. This can be a byte[] or a String. If
it is a String, the UTF-8 bytes will be used.
 * @return The message digest as a HEX encoded string
 * @throws UnsupportedEncodingException If UTF-8 is not supported (unlikely)
 * @throws NoSuchAlgorithmException If the requested algorithm is not supported
 */
public String getStringDigest (String algorithm, Object content) throws
UnsupportedEncodingException, NoSuchAlgorithmException;

```

IP Address Tool

The IP Address Tool allows the template to work with IP addresses as the following example shows:

```
##
## Show how to use the builtin content variable $ipAddress
##

#param($ip, 'ip', 'in', 'prompt=Enter an IP address')
#param($hostName, 'string', 'out')
#param($localhostName, 'string', 'out')
#param($localhostAddress, 'string', 'out')
#mockdevice(true)

## The tool $ipAddress is always available in a template

## The object returned by create(), createIPv4(), and createIPv6() is a java
InetAddress
## If the address cannot be created, the template will be halted
#set($inetAddress = $ipAddress.create($ip))
#set($hostName = $inetAddress.hostName)

#set($localhost = $ipAddress.localHost)
#set($localhostName = $localhost.hostName)
#set($localhostAddress= $localhost.hostAddress)
## Other methods in $ipAddressTool
## boolean isIp(Object)
## boolean isIPv4(Object)
## boolean isIPv6(Object)
## byte[] stringToIPv6Address(String)    ## Useful with Alteon
## String addressToString(byte[])    ## Useful with Alteon
```

The IP Address Tool has the following methods:

```
/**
 * Determine if the specified object represents an IP version 4 address.
 * @param address Any object
 * @return true if the object is an IPv4 address
 */
public boolean isIPv4 (Object address)

/**
 * Determine if the specified object represents an IP version 6 address.
 * @param address Any object
 * @return true if the object is an IPv6 address
 */
public boolean isIPv6 (Object address)

/**
 * Determine if the specified object represents an IP address.
 * @param address Any object
 * @return true if the object is an IP address
 */
```

```
(continued)

public boolean isIp (Object address)

/**
 * Converts a string representation of an IP address to a 16 byte address
 * array.
 * If the address is IPV4 the return value will be an IPV4-mapped IPV6 address
 * ::FFFF:a.b.c.d
 * @param ipAddress A string IP address
 * @return A 16 byte array
 */
public byte[] stringToIPv6Address (String ipAddress)

/**
 * Convert an address to a string representation.
 * Works for either 4 byte or 16 byte addresses.
 * @param address The address to convert
 * @return The string representation of the address
 */
public String addressToString (byte[] address)

/**
 * Create an InetAddress from the object specified
 * which should be an IP address, a host name, or an InetAddress.
 * @param address The host address or name
 * @return An InetAddress
 */
public InetAddress create (Object address)

/**
 * Create an InetAddress object from an IP v4 address.
 * @param address Any object
 * @return An InetAddress object or null if the object is not a valid IP
 * address
 */
public InetAddress createIPv4 (Object address)

/**
 * Create an InetAddress object from an IP v6 address.
 * @param address Any object
 * @return An InetAddress object or null if the object is not a valid IP
 * address
 */
public InetAddress createIPv6 (Object address)

/**
 * Get the local host IP address.
 * @return An InetAddress representing the local host
 */
public InetAddress getLocalHost ()
```

PEM Tool

The PEM Tool allows the template to inspect certificates, private keys, and certificate requests in PEM format. The tool provides different methods for each type of PEM file.

The following shows how to use the tool with a certificate:

```
## Example $pem tool with certificate

#mockdevice(true)
#param($certificatePath, 'string', 'in', 'format=pem', 'prompt=Path to
certificate PEM file')
#param($subjectDN, 'string', 'out')
#param($issuerDN, 'string', 'out')
#param($notAfter, 'string', 'out')
#param($notBefore, 'string', 'out')
#param($version, 'string', 'out')
#param($alternateNames, 'string', 'out')

#set($cert = $pem.parseCertificate($certificatePath))
#set($subjectDN = $cert.subjectDN)
#set($issuerDN = $cert.issuerDN)
#set($notAfter = $cert.notAfter)
#set($notBefore = $cert.notBefore)
#set($version = $cert.version)
#set($alternateNames = $cert.alternateNames)
```

The following example shows how to use the tool with a private key:

```
## Example $pem tool with private key

#mockdevice(true)
#param($privateKeyPath, 'string', 'in', 'format=pem', 'prompt=Path to private
key PEM file')
#param($password, 'string', 'in', 'prompt=Password for encrypted private key
file')
#param($validPassword, 'string', 'out')

#set($privateKey = $pem.parsePrivateKey($privateKeyPath))
#set($validPassword = $privateKey.isValidPassword($password))
```

The following example shows how to use the tool with a certificate request:

```
## Example $pem tool with certificate request

#mockdevice(true)
#param($certificateRequestPath, 'string', 'in', 'format=pem', 'prompt=Path to
certificate request PEM file')
#param($certificatePath, 'string', 'in', 'format=pem', 'prompt=Path to
certificate PEM file')
#param($validSignature, 'string', 'out')

#set($certReq = $pem.parseCertificateRequest($certificateRequestPath))
#set($cert = $pem.parseCertificate($certificatePath))
#set($validSignature = $certReq.isValidSignature($cert))
```

The tool has three main methods for parsing PEM data. Each method returns an object specific for the type of object parsed.

```
/**
 * Get a certificate object from an input source
 * @param pem can be a File, InputStream, Reader or String instance
 * @return a certificate object, null if source is not a certificate or cannot
be
 * parsed
 */
public Certificate parseCertificate(Object pem)

/**
 * Get a private key object from an input source
 * @param pem can be a File, InputStream, Reader or String instance
 * @return a private key object, null if source is not a private key or cannot
be
 * parsed
 */
public PrivateKey parsePrivateKey(Object pem)

/**
 * Get a certificate request object from an input source
 * @param pem can be a File, InputStream, Reader or String instance
 * @return a certificate request object, null if source is not a certificate
request
 * object or cannot be parsed
 */
public CertificateRequest parseCertificateRequest(Object pem)
```


The Certificate object returned by `$pem.parseCertificate()` has the following methods:

```
/**
 * Get a Date object indicating when this certificate ceases to be valid
 * @return Date
 */
public Date getNotAfter()

/**
 * Get a Date object indicating when this certificate begins to be valid
 * @return Date
 */
public Date getNotBefore()

/**
 * Get the DN of the subject of this certificate
 * @return String representing a distinguished name
 */
public String getSubjectDN()

/**
 * Get the DN of the issuer of this certificate
 * @return String representing a distinguished name
 */
public String getIssuerDN()

/**
 * Get the version of this certificate
 * @return an integer 1,2 or 3 representing the version of this cert
 */
public int getVersion()

/**
 * Get a list of string representations of the alternate names specified for
this certificate
 * @return a list of strings
 */
public List<String> getAlternateNames()
```

The CertificateRequest object returned by `$pem.parseCertificateRequest()` has the following methods:

```
/**
 * Indicates if signature in this request matches public key of provided
certificate
 * @param cert certificate containing public key
 * @return boolean
 */
public boolean isValidSignature(Certificate cert)
```

Subnet Tool

The Subnet Tool allows the user to access and use subnet utility methods:

```
#mockdevice(true)

#if($subnet.isValidAddressAndMask("192.168.1.1","255.255.255.0"))
    $subnet.create("192.168.1.1","255.255.255.0")
    Using Subnet tool (1):
    $subnet.networkAddress
#end

#if($subnet.isValidAddressAndMask("not valid","255.255.255.0"))
    $subnet.create("not valid","255.255.255.0")
    Using Subnet tool (2):
    $subnet.networkAddress
#else
    Invalid ..
#end
```

The following examples show how to work with the Subnet Tool:

```
public SubnetTool()

/**
 * A create method that takes a CIDR-notation string, e.g. "192.168.0.1/16"
 * <b>Should be called before any other method is used (except the isValid
 * methods)</b>
 *
 * @param cidrNotation
 *         A CIDR-notation string, e.g. "192.168.0.1/16"
 * @return A new subnet tool
 * @throws IllegalArgumentException
 *         if the parameter is invalid, i.e. does not match n.n.n.n/m
 *         where n=1-3 decimal digits, m = 1-3 decimal digits in range
 *         1-32
 */
public SubnetTool create(String cidrNotation)

/**
 * A create method that takes a dotted decimal address and a dotted decimal
 * mask. <b>Should be called before any other method is used (except the
 * isValid methods)</b>
 *
 * @param address
 *         An IP address, e.g. "192.168.0.1"
 * @param mask
 *         A dotted decimal netmask e.g. "255.255.0.0"
 * @return A new subnet tool
 * @throws IllegalArgumentException
 *         if the address or mask is invalid, i.e. does not match
 *         n.n.n.n where n=1-3 decimal digits and the mask is not all
 *         zeros
 */
```

```
(continued)

public SubnetTool create(String address, String mask)

/**
 * A validation method that takes a CIDR-notation string, e.g.
 * "192.168.0.1/16"
 *
 * @param cidrNotation
 *         A CIDR-notation string, e.g. "192.168.0.1/16"
 * @return true if valid
 */
public boolean isValidCidrNotation(String cidrNotation)

/**
 * A validation method that takes a dotted decimal address and a dotted
 * decimal
 *
 * @param address
 *         An IP address, e.g. "192.168.0.1"
 * @param mask
 *         A dotted decimal netmask e.g. "255.255.0.0"
 * @return true if valid
 */
)
public boolean isValidAddressAndMask(String address, String mask)

/**
 * Returns true if the parameter <code>address</code> is in the range of
 * usable endpoint addresses for this subnet. This excludes the network and
 * broadcast addresses.
 *
 * @param address
 *         A dot-delimited IPv4 address, e.g. "192.168.0.1"
 * @return True if in range, false otherwise
 */
public boolean isInRange(String address)

public String getBroadcastAddress()

public String getNetworkAddress()

public String getNetmask()
```

```
(continued)

public String getAddress()

/**
 * Return the low address as a dotted IP address. Will be zero for CIDR/31
 * and CIDR/32 if the inclusive flag is false.
 *
 * @return the IP address in dotted format, may be "0.0.0.0" if there is no
 *         valid address
 */
public String getLowAddress()

/**
 * Return the high address as a dotted IP address. Will be zero for CIDR/31
 * and CIDR/32 if the inclusive flag is false.
 *
 * @return the IP address in dotted format, may be "0.0.0.0" if there is no
 *         valid address
 */
public String getHighAddress()

/**
 * Get the count of available addresses. Will be zero for CIDR/31 and
 * CIDR/32 if the inclusive flag is false.
 *
 * @return the count of addresses, may be zero.
 */
public int getAddressCount()

public int asInteger(String address)

public String getCidrSignature()

public String[] getAllAddresses()
}
```

CHAPTER 6 – CONFIGURATION TEMPLATES DIRECTIVE REFERENCE

This section describes the following directives:

- [Answer Directive, page 85](#)
- [DefensePro Paste Mode Directive, page 87](#)
- [Device Directive, page 87](#)
- [DeviceApi Directive, page 89](#)
- [Device Lock Directive, page 89](#)
- [Error Directive, page 90](#)
- [Expect Directive, page 90](#)—These directives are not part of standard Velocity, but are Radware extensions to the Velocity template language for use with configuration templates.
- [Log Directive, page 92](#)
- [Mock Device Directive, page 92](#)
- [Parameter Directive, page 93](#)
- [Property Directive, page 95](#)
- [Result Directive, page 96](#)
- [Save Replies Directive, page 98](#)
- [Select Directive, page 98](#)
- [Timer Directive, page 99](#)
- [Typedef Directive, page 99](#)

Answer Directive

Alteon sometimes responds to a command in a configuration template with a question that must be answered before the next command can be sent. By default, vDirect can automatically detect when such questions are asked and answers with a “y” response.

The template developer can use the `#answer` directive to control if and how vDirect answers such questions. This directive is a block Velocity directive that must be terminated by `#end` in the template. The answer policy is in effect during execution of any context inside the directive. When `#end` is encountered, the previous answer policy is restored. This directive can be nested.



Note: If the template uses the `#savereplies` directive to disable Alteon response processing, no questions can be answered automatically. In this case, the template must always include answers to any questions in the template source, regardless of the current answer policy.

The available forms of the `#answer` directive are described below.

Disable or Enable Automatic Answers

`#answer(false) ... #end` disables automatic answers to questions. In this case, the template must include any required answers in its source. If you know that a certain Alteon command will always result in a question, you can disable automatic answers and insert the answers you want directly in the template:

```
#answer(false)
    /oper/slb/dis 1
    n
    n
#end
```

`#answer(true) ... #end` enables default automatic answers to questions. The engine answers “y” to any question it identifies. Since this is the default behavior, it would only ever be needed inside a nested `#answer` block.

Change The Automatic Answer

`#answer('n') ... #end` changes the automatic answer to “no”. Radware recommends that you use the value “n”.

```
#answer('n')
    /oper/slb/dis 1
    /* vDirect will automatically answer the questions with 'n'
#end
```

`#answer('y') ... #end` changes the automatic answer to “yes”. This is equivalent to writing `#answer(true)`. Radware recommends that you use the value “y”.

Use a List of Answers

`#answer(firstAnswer, secondAnswer, ...) ... #end` uses the answers specified exactly once, and in the order specified. When the list is exhausted, the last value is used to answer any additional questions. The list may alternatively be supplied as a Velocity array.



Note: The list is not shared across multiple Alteon devices. Each device gets exactly one copy of the list.

```
#answer('n', 'y')
    /* The first question will be answered with 'n'
    /* and all subsequent answers will be 'y'
    /oper/slb/dis 1
#end
```

Use a Map of Answers

`#answer({question1 : answer1, question2 : answer2, ...}) ... #end` specifies a map of regular expressions, and the answer that should be used with each. If one of the regular expressions is found in the question asked by the Alteon, its corresponding answer is used. If none of the expressions match, a “y” answer is supplied. If multiple expressions match the question, it is undefined which answer is used.

```
#answer({'Allow cookie': 'n', 'Mark existing': 'y'})
    /oper/slb/dis 1
#end
```

DefensePro Paste Mode Directive

Use the `#dpWithPasteMode` block directive to turn paste mode on or off for a DefensePro device. This directive can be used with the `pasteMode` `#device` attribute for a DefensePro for controlling when paste mode is used, or to prevent its use entirely.

```
#dpWithPasteMode ( $defensePro, enable) ... #end
```

The first argument must be a DefensePro device. The second argument must be a Boolean expression. This directive can be safely nested, and since the state is tracked during the entire CLI session with the DefensePro, no actual commands are issued unless needed to achieve the desired paste mode state.

Device Directive

Use the `#device` directive to declare a required device connection for the template. The device is given a name, and a corresponding device variable is created with the same name.

The syntax of the `#device` directive is:

```
#device ( $name [, 'prompt = display_prompt'] [, 'type = alteon|defensePro']  
[, 'driverVersion = version'], ['deviceVersion = version'], ['lazyConnect =  
true'])
```

[Table 7 - Device Directive Attribute Definitions, page 87](#) describes the `#device` directive attributes.

Table 7: Device Directive Attribute Definitions

Attribute	Description
\$name	The variable name for the device. Mandatory.
deviceVersion	<p>The actual version of the device.</p> <p>For example:</p> <ul style="list-style-type: none"> For version 8 and later (version 8.x), set <code>deviceVersion=8.*</code> For versions 8, 9, and 10 and later (versions 8.x, 9.x and 10.x), set <code>deviceVersion=8.*,9.*,10.*</code> <p>To specify all versions, leave the parameter empty.</p> <p>Note: vDirect will interrogate the device before executing the template or displaying the selection list to the user to ensure that this requirement is met.</p>
driverVersion	<p>The version of the device driver that vDirect should use, regardless of the actual version of the device. Optional.</p> <p>If the template has been written using lots of factory beans from a specific device driver version, it may be helpful to specify this version.</p>
lazyConnect	A Boolean attribute that can be set to delay any attempt to establish a CLI connection to the device until it is actually required by the template. Optional.
log	When set to true, the value of this parameter is included in the vDirect audit log.
maxLength	The maximum allowed length of an array when the parameter is an array.
minLength	The minimum allowed length of an array when the parameter is an array.

Table 7: Device Directive Attribute Definitions

Attribute	Description
pasteMode	For DefensePro only. When set to false, the template runs without ever enabling paste mode on DefensePro.
prompt	A string that displays when users select the device.
type	The type of device—alteon or defensePro. Appending [] to the type makes the parameter an array of devices.



Note: By convention, the name for an Alteon device is \$adc. The name for a DefensePro device is \$defensePro. However, vDirect does not require these names.



Note: If your template does not include any CLI commands, Radware recommends that you use `lazyConnect=true`. In some cases, configuration templates are used as a convenient way to author some device logic that does not actually require a CLI connection, which is an expensive overhead when it is not needed. If `lazyConnect` is true, no CLI connection will be established until and unless one of the following occurs:

- The template calls a device API that requires a CLI connection.
- The template generates any non-whitespace output that should be sent to the device.

If no `#device` directive is present in the template, the behavior is as if the following directive were used:

```
#device($adc, 'type=alteon')
```

Adding any `#device` directive overrides this default behavior. If the optional `deviceVersion` is specified, vDirect attempts to choose a device driver that matches the specified version, and also validates that the actual device supplied to the template meets the version requirement.

This example declares that the template can only execute against a DefensePro running version 7 of the DefensePro software:

```
#device($defensePro, 'type=defensePro', 'version=7.40')
```

You can declare an array of multiple device connections for a template by adding [] to the `#device` type, and an optional minimum and maximum array length can be specified:

```
#device($alteons, 'type=alteon[]', 'minLength=2', 'maxLength=10')
```

In this case, the Velocity context variable `$alteons` is an array. Each element of the array is the same `DeviceAdaptor?` that is normally used in configuration templates:

```
#foreach($alteon in $alteons)
#select($alteon)
/diff
#set($x = $alteon.readBean('/info/swkey'))
#set($dummy = $out.add($x.agEnabledSwFeatures))
/* $x.agEnabledSwFeatures
#end
```


DeviceApi Directive

Use the `#deviceapi` directive before using any device variable methods that may change the device configuration. These methods are listed at [Device Variables, page 63](#). This directive is especially important for DefensePro since attempting to change the configuration using the vDirect Device SDK while running a template may cause errors. To use the `#deviceapi` directive, wrap the Device SDK calls in a `#deviceapi($device) ... #end` block.

The syntax of the `#deviceapi` directive is:

```
#deviceapi($device)  template_code  #end
```

The following example shows how to use `#deviceapi` to safely call the delete and create methods of the device adapter:

```
device($defensePro, 'type=defensePro', 'version=7.40')

#param($name, 'prompt=Port name')
#param($portIndex, 'int', 'prompt=Port index')

#deviceapi($defensePro)
  #set($pPort = $defensePro.newBean('RsBWMPPhysicalPortGroupEntry'))
  #set($pPort.name = $name)
  #set($pPort.port = $portIndex)

  #if($defensePro.exists($pPort))
    ## Port exists
    #set($result = $defensePro.delete($pPort))
  #else
    ## Creating Port
    $defensePro.create($pPort)
  #end
#end

$defensePro.apply()
```



Note: It is not necessary to wrap calls to `apply()` or `commit()` with `#deviceapi`.

Device Lock Directive

By default, vDirect acquires a device lock for each Alteon accessed by a template. Waiting for the locks can block execution of the template. In addition, because vDirect does not automatically apply device changes made by templates, Alteon remains locked by the vDirect session that executed the template until the user or program applies the changes, reverts the changes, or explicitly unlocks Alteon.

You can use the `#devicelock` directive to execute the template without attempting to acquire device locks. Place a single `#devicelock` directive near the top of the template, and set it to `false`:

```
#devicelock(false)
```

Error Directive

The template can choose to halt execution with an error by using the `#error` directive. If this directive is used, the caller of the template receives an exception with the message text specified in the `#error` directive.

The syntax of the `#error` directive is:

```
#error (message)
```

Expect Directive

The `expect` feature allows the template developer to take full control of the CLI session with a device.

Except for `#expectBefore` and `#expectAfter`, the `expect` directives can only be used within a `#savereplies(false)` block since it is only then that vDirect automatic processing of CLI commands is disabled.

The `expect` feature is a very close match for the UNIX `expect` tool. It is the responsibility of the template developer to issue the correct `expect` commands to fully process the expected responses from the device. If responses are left unconsumed, it can be difficult or impossible for vDirect to resume automatic response processing.

[Table 8 - Expect Directives, page 90](#) describes `expect` feature directives.

Table 8: Expect Directives

Directive	Description
<code>#expect(regex)</code>	A line directive that takes a regular expression and waits until the input matches. Note: The pattern can be omitted to evaluate the before and after patterns.
<code>#expectCase()</code>	A block directive that includes one or more <code>#case</code> directives as children. <code>#expectCase</code> waits for input to match one of the <code>#case</code> directives, and then executes the matching <code>#case</code> . The regular expressions are evaluated in the order listed. An <code>#expectCase</code> block with no <code>#case</code> directives can be used to evaluate the before and after patterns.
<code>#case(regex)</code>	Use this block directive inside <code>#expectCase</code> to match the specified pattern.
<code>#caseTimeout()</code>	Use this block directive inside <code>#expectCase</code> to catch the case where no match is found within the timeout period. If a <code>#caseTimeout</code> is provided, the value of <code>\$expect.failOnTimeout</code> is ignored.
<code>#caseEOF()</code>	Use this block directive inside <code>#expectCase</code> to catch the case when the device connection is closed or fails. If a <code>#caseEOF</code> is provided, the value of <code>\$expect.failOnEOF</code> is ignored.

Table 8: Expect Directives

Directive	Description
<code>#expectContinue()</code>	Use this line directive inside <code>#case</code> to re-evaluate the entire enclosing <code>#expectCase</code> block. This is a power feature that lets you process prompts and continue waiting for a response.
<code>#expectBefore()</code>	<p>Identical to <code>#expectCase()</code> except that no action is taken. Instead, a set of before patterns are installed for expect. These have the same meaning as before patterns in UNIX expect.</p> <p>You cannot use <code>#caseTimeout</code> and <code>#caseEOF</code> inside <code>#expectBefore</code>. Use an empty <code>#expectBefore()</code> <code>#end</code> block to remove any before patterns.</p>
<code>#expectAfter()</code>	<p>Identical to <code>#expectCase()</code> except that no action is taken. Instead, a set of after patterns are installed for expect. These have the same meaning as after patterns in UNIX expect.</p> <p>You cannot use <code>#caseTimeout</code> and <code>#caseEOF</code> inside <code>#expectAfter</code>. Use an empty <code>#expectAfter()</code> <code>#end</code> block to remove any after patterns.</p>

\$expect Context Variable

The `$expect` context variable is included in the Velocity context. It has the following properties:

Table 9: \$expect Properties

Property	Description
<code>buffer</code>	This string contains all of the unconsumed device output, including the last match up to and including the matched pattern. This property corresponds to UNIX <code>expect \$expect_out(buffer)</code> .
<code>input</code>	This string contains all of the unconsumed device output, including any character before and after the matched pattern.
<code>failOnTimeout</code>	Set this boolean property to true to cause an exception that halts the template if a pattern is not matched within the specified timeout. Default is true.
<code>failOnEOF</code>	Set this boolean property to true to cause an exception that halts the template if the device connection is closed while matching a pattern. Default is true.
<code>timeout</code>	<p>Set this property to the desired expect timeout in seconds. The default value is 10 seconds.</p> <p>Note: This property affects all subsequent device replies in the template, not only the replies in <code>#expect</code> directives. The developer can use this variable to set the reply timeout even if no <code>#expect</code> directives are used in the template.</p>
<code>timedOut</code>	Read this boolean property after a call to expect to determine if the previous expect timed out. Only available if <code>failOnTimeout</code> is set to false.
<code>groupCount</code>	Read this integer property to determine the number of groups in the regular expression that was last matched.
<code>groups</code>	Returns an array of groups in the regex that was last matched. Each object is a <code>MatchResult?</code> For more information, see MatchResult, page 92 .

Table 9: \$expect Properties

Property	Description
[] operator	Use Velocity's array index operator to directly reference a captured group. The object returned is a <code>MatchResult?</code> For more information, see MatchResult, page 92 . Group 0 refers to the entire regular expression. The expression <code>\$expect[0].string</code> corresponds to UNIX <code>expect_out(0,string)</code> .

MatchResult

Each `MatchResult?` object has the following read-only properties:

- `string`—The string that matches the captured group.
- `start`—The first index in the buffer of the captured group.
- `end`—The last index in the buffer of the captured group.

Log Directive

Template authors can use the `#log` directive to include messages in the vDirect logs.

The syntax of the `#log` directive is:

```
#log ( message [, "debug" | "info" | "warn" | "error"] )
```



Note: If the template is running as part of a vDirect workflow, each log message appears as an entry in the audit log of the workflow.

Mock Device Directive

Every configuration template requires at least one Alteon device when executed. By default, a single device named "adc" is required as part of the input parameters to the template if no `#device` directives are used. vDirect must be able to make a Telnet or SSH connection to this device, and the Alteon CLI commands contained in the template are executed against this device. The `#mockdevice` directive also supports running a template without connecting to any actual Alteon device. Template developers may insert the following directive in a template:

```
#mockdevice(true)
```

When used, each named device required by the template is replaced by a mock device adapter that simulates an Alteon device, but does not actually connect to any real device. In addition, any devices provided in the input parameters are optional and will be ignored, since each one is simply replaced by a mock device.

The `#mockdevice` directive supports the following use cases:

- Testing a template during development without the need to connect to any real Alteon device. The developer can set the argument to `#mockdevice` to true or false at any time during development.
- Using a configuration template to generate a CLI script for an unknown device. The configuration template can generate an Alteon CLI script that is saved and then later executed by the program against some real device. By specifying `#mockdevice(true)`, the program executing the template is never required to specify any device in the input parameters to the template, and can simply save the generated script for later use.

Parameter Directive

Templates may require input parameters, and can return output parameters. These parameters must be defined in the template using the `#param` directive.



Note: Radware strongly recommends that all parameter declarations are placed near the top of the template source file.

The syntax of the `#param` directive is:

```
#param($name [, "option = value"]*)
#param($timeWindow, 'type=int','direction=in','prompt=Certificate Expiration
Window (Days)','min = 1', 'max = 3650', 'properties={"password" :
true,"tooltip" : "someToolTip","maxCharLength" : "12"}')
```

[Table 10 - Template Parameter Definitions, page 93](#) displays the template parameter definitions.



Note: Attributes in [Table 10 - Template Parameter Definitions, page 93](#) with a “ui-” prefix are evaluated only by the client Web user interface, and are never evaluated by the vDirect server. The attribute value must be a Javascript expression, not a Velocity expression.

Table 10: Template Parameter Definitions

Option	Description
defaultValue	A default value for the parameter.
direction	<p>Specifies whether the parameter is an input parameter, output parameter, or both.</p> <p>Values:</p> <ul style="list-style-type: none"> • in—The parameter is input only. • out—The parameter is output only. • inout—The parameter is both input and output. <p>Default: inout</p>

Table 10: Template Parameter Definitions

Option	Description
format	Specifies an optional format specifier for string parameters. Values: <ul style="list-style-type: none"> multiline—The string value is multiline. pem—The string value is in PEM file format. vDirect validates that the value is a valid PEM format. password—The string parameter should be displayed as a password. html—The string parameter should be displayed as HTML.
log	When set to true, the value of this parameter is included in the vDirect audit log.
max	Specifies an optional maximum allowed value for int parameters.
maxCharLength	The maximum allowed length of a string value.
maxLength	The maximum allowed length of an array if the parameter is an array type.
min	Specifies an optional minimum allowed value for int parameters.
minCharLength	The minimum required length of a string value.
minLength	The minimum allowed length of an array if the parameter is an array type.
name	Specifies the parameter name. Mandatory.
pattern	Specifies an optional regular expression that can be used with string parameters. When validating the parameter, its value must match the regular expression.
prompt	A description of the parameter that is displayed to an end-user running the template.
properties	An opaque string value associated with the parameter. vDirect does not use this value, but it may be used by a client UI to help display the parameter.
separator	Specifies a separator to be used for delimiting array values. The default is a comma.
type	Specifies the parameter type. For more information, see Input and Output Parameter Types, page 133 .
uiEditable	A Javascript expression that returns a Boolean value that determines if the user can edit this parameter.
uiRequired	A Javascript expression that returns a Boolean value that determines if a value is required for this parameter. Note: When uiRequired is specified, required must also be set to false.
uiTooltip	A Javascript expression that calculates the tooltip for the input field.
uiVisible	A Javascript expression that returns a Boolean value that determines if the input field is visible.
values	Specifies an optional set of comma-separated choices for the parameter. When validating the parameter, its value must be one of the choices. This option can be used with the string and int parameters.

Table 10: Template Parameter Definitions

Option	Description
valuesOnly	<p>This boolean option is only valid when the <code>values</code> option is also specified. If <code>valuesOnly</code> is set to <code>false</code>, the server does not require that the parameter value is one of the values specified by the <code>values</code> option.</p> <p>Values:</p> <ul style="list-style-type: none"> <code>true</code>—The vDirect Web user interface displays drop-down lists for users to select a value. <code>false</code>—The vDirect Web user interface displays editable combobox so that users may either choose one of the values, or type in a different value. <p>Default: <code>true</code></p>

A shorter, alternative syntax for parameter definitions, which lets you specify the name, type, and direction as the first three arguments in the parameter definition without including the name for each option, is also supported:

```
#param($name, "type", "direction", [, "option = value"]*)
```

The following is an example of defining template parameters using this syntax:

```
#param($vipAddr, "ip", "in", "prompt=IP Address for the Virtual Service")
#param($srvIps, "ip[]", "in", "prompt=Comma Delimited List of Server IPs")
#param($complvl, "int", "in", "min=1", "max=9", "prompt=Compression")

#param($comppolId, "string", "out")
#param($virtId, "int", "out")
#param($groupId, "int", "out")

#param($timeWindow, "int", "in", "prompt=Certificate Expiration Window
(Days)", "min = 1", "max = 3650", "properties={"password" : true,
"maxCharLength" : "12"}")
```

When running the template, all the input parameters declared are available to the template as Velocity context variables, using the names of the parameters.

Property Directive

The `#property` directive can be used to store additional meta-data about the template. vDirect does not use this information but it is available to external applications that may make use of it. For example, for the vDirect command line console to display the content of the property named "description".

The syntax of the `#property` directive is:

```
#property( "name", "value")
```

Result Directive

In addition to setting output variables, the configuration template can also return a human-readable result using the `#result` block directive. The entire child contents within the `#result` directive are simply rendered and returned as the result.

The syntax of the `#result` directive is:

```
#result ( mediaType ) ... #end
```

The `#result` directive takes an optional additional argument that is the suggested filename of the result when downloading. The argument can be an expression:

```
#result('text/plain', 'example.txt')
```

The following template creates an HTML report of the current port status for an Alteon:

```
#property('title', 'Alteon port link status report')  
#property('description', 'Create a report of Alteon link status')  
#property('tags', 'alteon, report')  
#property('icon', 'Photoshop.png')
```



```
(continued)

#device($adc, 'type=alteon', 'prompt=Alteon')

## The output could be used programatically
#param($output, 'type=object', 'direction=out')

#set($portInfo = [])

#set($management = $adc.readBean('AgMgmtPort'))
#set($managementPort = $adc.readBean('MgmtInfo'))
#set($info = {})
#set($info.speed = "$managementPort.mgmtPortInfoSpeed")
#set($info.duplex = "$managementPort.mgmtPortInfoMode")
#set($info.flowControl = "$management.agMgmtPortNewCfgMode")
#set($info.link = "$managementPort.mgmtPortInfoLink")
#set($info.status = 'UP')
#set($info.name = "management")
#set($info.description = "")
#set($dummy = $portInfo.add($info))

#set($ports = $adc.readAllBeans('PortInfoTableEntry'))
#foreach($port in $ports)
    #set($info = {})
    #set($info.speed = "$port.speed")
    #set($info.duplex = "$port.mode")
    #set($info.flowControl = "$port.flowCtrl")
    #set($info.link = "$port.link")
    #set($info.status = "$port.phyIfOperStatus")
    #set($info.description = $port.phyIfDescr)
    #set($info.name = $port.indx)

    #set($dummy = $portInfo.add($info))
#end

#set($output = $portInfo)

#result('text/html')

<div>

<center><h3>Ports for Alteon ${adc.ip}</h3></center>
<br/>

<table border="1">

<tr>
<th>Port</th>
<th>Speed</th>
<th>Duplex</th>
<th>Flow Control</th>
<th>Link Status</th>
<th>Operator Status</th>
<th>Description</th>
</tr>
```

```
(continued)

#foreach($port in $portInfo)

<tr>
<td>${port.name}</td>
<td>${port.speed}</td>
<td>${port.duplex}</td>
<td>${port.flowControl}</td>
<td>${port.link}</td>
<td>${port.status}</td>
<td>${port.description}</td>
</tr>

#end

</table>
</div>

#end
```

Save Replies Directive

The `#savereplies` directive improves template performance by removing the requirement to read each reply from Alteon before moving to the next command. This behavior remains in effect until the enclosing `#end` directive is encountered. vDirect behavior then returns to the state defined before `#savereplies`.

```
#savereplies(boolean arg)
##some template code
#end
```

Select Directive

The `#select` directive is required when the template has specified more than one `#device` directive. Use the `#select` directive to choose which device the template content is sent to.

The syntax of the `#select` directive is:

```
#select($device)
```

The selected device remains in effect until another `#select` directive is processed. All CLI commands following a `#select` directive are sent to the same device.

Timer Directive

The `#timer` block directive can be used during development to time execution. The directive outputs to the log the execution time for the block of code enclosed by the directive.

The syntax of the `#timer` directive is:

```
#timer ( 'comment' ) code #end
```

Typedef Directive

In addition to the predefined parameter types listed for the `#param` directive, vDirect also supports user defined, complex parameters types that are composed from one or more fields.

Use the `#typedef` block directive to define user types in the configuration template. Types must be defined before they are used in a `#param` directive:

```
#param($name [, "option = value"]*)
```

The following example defines an object with four fields named `id`, `color`, `delay`, and `isl`:

```
#typedef('myType')
  #param($id, 'prompt = UUID')
  #param($color, 'prompt = Color', 'values = red,green,blue')
  #param($delay, 'int', 'prompt = Delay', 'defaultValue = 0', 'min = 0')
  #param($isl, 'islId', 'defaultValue=1/1')
#end
```

The fields of a user defined type are defined using the same `#param` directive used for defining parameters to the template. However, when used within a `#typedef` directive, the direction property of `#param` has no meaning and is ignored.

Once defined, the new type may be used in regular `#param` directives just like any other type:

```
#param($var1, 'myType', 'inout')
```

Default Values

Like other parameters, the template may define a default value for a user-defined parameter type. The syntax for the default value must be a JSON object:

```
#param($var1, 'myType', 'in', 'defaultValue={"id": "Name1", "color": "red"}')
```



Note: In the example, it is optional to assign values to the `delay` and `isl` fields because default values for those fields are defined within the `#typedef` itself. Default values defined within the `#param` override default field values specified in the `#typedef` directive.

If the `#typedef` specifies a default value for every field, the following can be used:

```
#param($var1, 'myType', 'in', 'defaultValue={}')

```

User Types as Fields

User-defined types may include fields using another user defined type. For example:

```
#typedef('typeA')
    #param($field1, 'int')
    #param($field2, 'string')
#end

#typedef('typeB')
    #param($typeA_1, 'typeA')
    #param($typeA_2, 'typeA', 'defaultValue = {"field1":22, "field2":
"Hello"}')
#end

#param($var1, 'typeB')
#param($var1, 'myType', 'in', 'defaultValue={}')

```

Accessing User Type Fields in the Template

In the configuration template, user-defined parameters are represented as Map objects. The fields may be accessed as though they are properties, or by using Velocity's indexed syntax. Setting an output value is shown in the following example:

```
#typedef('myType')
    #param($id, 'prompt = UUID')
    #param($color, 'prompt = Color', 'values = red,green,blue')
#end

#param($var1, 'myType', 'in')
#param($var2, 'myType', 'out')

/* The value of var1.color is: $var1.color */
/* The value of var1.id is: $var1['id'] */

#set($var2 = {"id" : "outId", "color" : "blue"})
#set($var2.color = "red")

```

Arrays of User Types

Whenever a user type defined is defined, its array type may also be used for a parameter that should be an array of objects:

```
#typedef('myType')
    #param($id, 'prompt = UUID')
    #param($color, 'prompt = Color', 'values = red,green,blue')
#end

#param($var1, 'myType[]', 'prompt = An array of myType', 'defaultValue=[]')

```

Java Support

In Java, user-defined parameters have the type `Map<String, Object>` as in the following example:

```
Map<String, Object> parameters = new HashMap<String, Object>();
Map<String, Object> myType = new HashMap<String, Object>();
myType.put("id", "Name");
myType.put("color", "red");
parameters.put("var1", myType);

AdcTemplateResult result = templates.debugConfigurationTemplate(
    "myTemplate.vm", parameters, connection);

System.out.println(((Map<String, Object>)
parameters.get("var2")).get("color"));
```

REST API Support

When passing template parameters using the REST API, user-defined types are simply JSON objects, so no special translation is required:

```
POST /api/workflow/ud/action/test HTTP/1.1
Content-Type: application/vnd.com.radware.vdirect.template-parameters+json
Cache-Control: no-cache

{
  "parameters": {
    "var1": { "id": "Rest", "color": "blue" }
  }
}
```


CHAPTER 7 – WORKFLOW TEMPLATE REFERENCE

This chapter provides an introduction to authoring workflow templates for vDirect. A workflow template includes a required workflow descriptor file named **workflow.xml**, and any additional files needed by the template. The additional files will include configuration templates and groovy scripts that are executed by workflow actions. These files are packaged together as a .zip archive file and uploaded to vDirect.



Note: The vDirect client package includes sample workflow templates that you can examine and use to develop your own workflows.

Workflow Element Definitions

The following workflow elements are used by developers:

- Workflow descriptor—Contained in the **workflow.xml** file in the template .zip archive. It defines all of the states, persistent parameters, actions, inputs, and steps of the workflow template.
- Sequence—An ordered set of steps that execute as part of an action.
- Step—A single step in a sequence can be one of several supported types of steps.
- Scripting Step—Configuration templates and groovy scripts are together called scripting steps. They provide the bulk of the functionality of the workflow template.
- Configuration templates—Can be bundled in the workflow template archive and executed by the configuration step in the workflow descriptor.
- Groovy script—Source files containing Groovy scripting commands can be bundled in the workflow template archive and executed by the script step in the workflow descriptor. Groovy scripts have access to the full vDirect SDK and can be used to perform virtually any function required by a workflow.
- Error handler—Sequences that are executed when actions encounter errors. They ensure that the workflow remains in a stable, known state.

Workflow Concepts

Workflows are based primarily on three concepts that are defined in the workflow descriptor—persistent properties and devices, states, and actions.

Persistent properties and devices are stored as part of the workflow and are passed into each action which can update their values. This feature allows workflows to be stateful over time. The outputs from running one action can later be used as inputs to another action. The properties that persist with the workflow must be explicitly defined in the workflow descriptor. Actions may use other input or calculated properties when they execute, but these will be local properties and will not persist with the workflow. The Alteon devices on which the workflow operates can be accessed as properties, but they are declared separately from the properties because vDirect provides special support for devices in actions.

States can be used in workflows to implement a state transition diagram that creates a life cycle for the workflow. States also control which actions can be executed based on the current state of the workflow. The state names "created," "deleted," "any," and "none" are predefined by vDirect and may not be used as user-defined states.

Actions are the basis of workflow functionality. Actions usually use vDirect configuration templates to configure Alteon devices, however actions can be used to accomplish almost any task that is possible using vDirect.

When a workflow is first created, default values for persistent properties are specified, and the state is "created." As the workflow executes, the values of the persistent parameters are updated, the current state is changed, and the workflow is saved in its new state.

Velocity Expressions

In the following sections, many attribute values allow for an expression in the workflow descriptor. These expressions are Velocity template expressions that have full access to the parameter and device variables in the action. The term "expression" for an attribute value refers to a Velocity template expression. For more information about the syntax of these expressions, see the Velocity User Guide at <http://velocity.apache.org/engine/devel/user-guide.html>.

The following shows examples of expressions in the workflow descriptor:

```
<!-- Simple variable reference -->
<deleteWorkflow workflow="$islWorkflow"/>
<!-- Conditional expression in a sequence element -->
<sequence if="{service.islId.anyVlan} == false">

<!-- Accessing the persistent properties of a child workflow -->
<map from="$l3Workflow['primaryClientAddress']" to="$peer1clientInterfaceIP"/>
```


Workflow Descriptor

The workflow descriptor must be located in the **workflow.xml** file in the root of the workflow .zip archive. The file must have the following structure:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="workflowName">

  <description>A brief description of the workflow</description>

  <persist>
    <!-- Declares the persistent devices associated with the workflow -->
    <devices>
      <!-- This example just uses a single ADC named adc -->
      <device name="adc"/>
    </devices>
    <!-- Declares the persistent parameters of the workflow -->
    <parameters>
      <parameter name="vipAddress" type="ip" prompt="VIP address"/>
      <parameter name="ServerIps" type="ip[]"/>
      <parameter name="virtId" type="int"/>
      <parameter name="complvl" type="string" defaultValue="1"/>
    </parameters>
  </persist>

  <!-- Declare the states used by this workflow -->
  <states>
    <state name="applied"/>
    <state name="removed"/>
  </states>

  <!-- Declare the workflow actions -->
  <actions>
    <action name="apply" fromState="removed" toState="applied">
      <inputs>
        <!-- These are the inputs collected from the user whenever this
action is run.

                They can be a mix of persistent parameters and "local"
                parameters for the action.
-->
        <devices>
          <device name="adc"/>
        </devices>
        <parameters>
          <parameter name="vipAddress"/>
          <parameter name="ServerIps"/>
          <parameter name="complvl"/>
        </parameters>
      </inputs>
      <sequence>
        <configuration name="setup" file="compression.vm"/>
      </sequence>
    </action>
```

```
<action name="remove" fromState="applied" toState="removed">
  <!-- Note that no additional inputs are needed to run the remove
action. -->
  <sequence>
    <configuration name="teardown" file="anticompression.vm"/>
  </sequence>
</action>
</actions>
</workflow>
```

The namespace attribute is required, as follows:

```
xmlns="http://www.radware.com/vdirect"
```

The descriptor defines the persistent parameters of the workflow, then the user-defined states used in the workflow, and finally each action available in the workflow.



Note: The vDirect client package includes the **workflow.xsd** file that fully documents the workflow descriptor file format. Consult the schema definition for the latest documentation on the available tags that can be used in the workflow descriptor.

The `<parameter/>` element in the workflow descriptor works just like the **#param** directive in configuration templates, and supports all of the same options. For more information, see [Table 10 - Template Parameter Definitions, page 93](#).

The `<device/>` element in the workflow descriptor works like the **#device** directive in configuration templates and supports the same options. If the device type is not specified, it is assumed to be Alteon. To use a DefensePro device with a workflow, use a declaration like this following:

```
<device name="defensePro" type="defensePro" version="7.40"/>
```

Although the examples in this chapter are for Alteon devices, workflows can use DefensePro devices in the same way.

User-defined types are also supported in workflows and must be defined in the **workflow.xml** file before the `<persist/>` element. The following example shows that user-defined types can be used both for persistent properties of the workflow, as well as for action local variables:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="userDefined">

  <typedef name="typeA">
    <parameter name="a" type="int" defaultValue="2"/>
    <parameter name="b" type="int[]" />
  </typedef>

  <typedef name="typeB">
    <parameter name="id" prompt = "UUID"/>
    <parameter name="typeA" type="typeA" defaultValue='{"b": [1,2,3]}' />
  </typedef>
```

```
<persist>
  <parameters>
    <parameter name="intArray" type="int[]" />
    <parameter name="var1" type="typeB" defaultValue='{ "id": "hm3" }' />
  </parameters>
</persist>

<actions>
  <action name="test">
    <inputs>
      <parameters>
        <parameter name="intArray" />
        <parameter name="var2" type="typeB" />
      </parameters>
    </inputs>
    <sequence>
      <log message="intArray = $intArray" />
      <log message="var1 = $var1" />
      <log message="var2.id = ${var2.id}" />
      <set saveAs="$var2.typeA.a" value="33" />
    </sequence>
  </action>
</actions>
</workflow>
```

Workflow Actions

Actions are the basis of workflow functionality. Actions can require user inputs, perform a sequence of steps, define error handlers, and update the workflow state.

When an action runs, the following happens:



Note: The action does not run if the current workflow state does not match the specified fromState of the action.

1. Required inputs defined by the action are validated. If any required inputs are missing the action does not run.
2. Local devices defined by the action are initialized.
3. A local action parameter scope is created for the action, and the current values of all workflow persistent parameters are copied into this scope, followed by any input parameters and local devices for the action. Finally, a special variable named \$workflow is added to the local parameters that provide access to information about the current workflow.
4. The sequence of steps for the action begins to run. All steps in the action share the same local parameter scope. Steps have access to the persistent parameters of the workflow, and can also share local variables whose lifetime is only the duration of the action.
5. If any step fails, the action halts and error handlers are executed. All changes to the workflow persistent parameters are undone, and the workflow is left unchanged (see [Error Handling, page 113](#)).
6. If all steps succeed, the workflow persistent parameters are updated, the new workflow state is set to the action's toState, and the workflow is saved.

[Table 11 - Action Element Attributes, page 108](#) lists the attributes for the action element in the workflow descriptor.

Table 11: Action Element Attributes

Attribute	Description
consentRequired	When set to false and the action has no inputs, the vDirect Web user interface allows users to run this action with a single click.
fromState	The allowed state for running this action. The default value is "any".
name	The required name of the action.
permission	The optional permission required by users to execute this action. If specified, this permission must be added to the user's role before they can run the action.
title	The title of the action displayed to users in the vDirect action catalog.
toState	The new workflow state after running this action. The default value is "any" which means the workflow state will remain unchanged.
visible	Optional Boolean value that hides this action from end users. This can be used for a createAction or deleteAction that you want to hide from users.

[Table 12 - Child Elements, page 108](#) lists child elements available to an action element.

Table 12: Child Elements

Element	Description
devices	Optionally defines local devices for the action (see Local Devices, page 112).
inputs	Defines the required user inputs for this action (see Inputs, page 108).
onError	Defines an error handler for the action (see Error Handling, page 113).
outputs	Optionally defines the required outputs for this action (see Outputs, page 110).
result	Optionally defines the result for this action (see Results, page 111).
sequence	Defines a sequence of steps that will be run (see Sequences, page 113).

Inputs

An action can optionally require inputs. Inputs can consist of properties and devices, and are specified by the input child element. The following shows how to define inputs for the action:

```
<action name="apply" fromState="removed" toState="applied">
  <inputs>
    <devices>
      <device name="adc"/>
    </devices>
    <parameters>
      <parameter name="vipAddress"/>
      <parameter name="ServerIps"/>
      <parameter name="complvl"/>
      <parameter name="localVariable" type="string" prompt="Enter a value"/>
    </parameters>
  </inputs>
  <sequence>
    <configuration name="setup" file="compression.vm"/>
  </sequence>
</action>
```

vipAddress, **ServerIps**, and **complvl** are persistent parameters in the workflow. It is not necessary specify parameter types or display prompts because vDirect gets these from the persistent parameter definitions. However, the **localVariable** parameter is not the name of a persistent parameter in the workflow, so additional information is supplied as part of the input parameter. **localVariable** will become a local action parameter and its lifetime will only correspond to the execution of the action.



Note: Local action parameters cannot hide or redefine a persistent workflow parameter with the same name.

XML developers can add a `<renderTable/>` child element to the `<parameter/>` element to specify that an array of a user-defined-type should be rendered in the UI using a table instead of a sequential list.

At least one column must be specified. The following boolean attributes can also be specified, and each is true by default:

- **creatable**—The user can create new rows in the table.
- **deletable**—The user can delete rows from the table.
- **editable**—The user can edit rows in the table.
- **reorderable**—The user can change the order of rows in the table.

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="test-simple-table">

  <typedef name="Foo">
    <parameter name="a" type="ipv4" prompt="Network Address"
defaultValue="172.20.20.0" />
    <parameter name="b" type="ipv4" prompt="Subnet Mask"
defaultValue="255.255.254.0"/>
  </typedef>

  <typedef name="Bar">
    <parameter name="foo" type="Foo[]" prompt="Foo Items">
      <renderTable creatable="true" deletable="true" editable="true"
reorderable="true">
```

```
(continued)

<column property="a" header="A"/>
  </renderTable>
</parameter>
</typedef>

<actions>
  <action name="init" runAs="sequence">
    <inputs>
      <parameters>
        <parameter name="x" type="Foo[]">
          <renderTable creatable="true" deletable="true"
editable="true" reorderable="true">
            <column property="a" header="A"/>
          </renderTable>
        </parameter>
      </parameters>
    </inputs>
    <runNext action="chainConfig" cancelAction="deleteWorkflow"/>
  </action>

  <action name="chainConfig" toState="deleted" visible="false">
    <inputs>
      <parameters>
        <parameter name="y" type="Bar"/>
      </parameters>
    </inputs>
  </action>
</actions>

</workflow>
```

Outputs

To add new output parameters, define the name and any other desired attributes of the parameter. In the following example, the outputs include the input parameter "a" and an integer parameter named "foo".



Note: The values of the output parameters are simply taken from the variable binding after the action completes. Use the `<set>` element to assign variables to the binding. The following shows how to define outputs for the action:

```
<action name="update">
  <inputs>
    <parameters>
      <parameter name="a"/>
    </parameters>
  </inputs>
  <outputs>
    <parameter name="a"/>
    <parameter name="foo" type="int"/>
  </outputs>
  <sequence>
    <set saveAs="$foo" value="$a % 10"/>
  </sequence>
</action>
```

Results

Independent of output parameters, the workflow action can also return a **result**. Since we expect the result to be consumed by human users, the expected content-type of the result is "text/plain" or "text/html", but there is nothing in the design that requires this.

To return a result from an XML workflow action, add a `<result>` element immediately after the last sequence, and before any error handlers:

```
<action name="update">
  <inputs>
    <parameters>
      <parameter name="a"/>
    </parameters>
  </inputs>
  <outputs>
    <parameter name="a"/>
    <parameter name="foo" type="int"/>
  </outputs>
  <sequence>
    <log message="Update $a"/>
    <set saveAs="$foo" value="$a % 10"/>
    <error if="$foo < 5" message="Failed this time"/>
  </sequence>
  <result type="text/plain" value="$foo"/>
</action>
```

The example above shows the simplest possible result in which an existing variable in the binding is returned as the result by setting the value attribute of the result element. The type property is required.

You can also generate text results using a plain velocity template:

```
<result type="text/plain" template="update-result.vm"/>
```

You can use the **filename** value control the suggested filename displayed when a user downloads the result. The value can be an expression:

```
<result type="text/plain" value="'This is the result'" filename="foobar.txt"/>
```



Note: This template **is not a configuration template**. There are no **#param** directives, or device connections. The rendered template is used to generate the result, not to send commands to a device. The template receives the entire current binding as its variable context.

Local Devices

Sometimes an action may interact with Alteon devices that are not stored as persistent devices in the workflow. Local devices can be defined for the action. These devices are initialized when the action runs and, like other local variables, they are forgotten when the action completes.

The following shows how to define local devices for the action:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="localDeviceExample">

  <persist>
    <parameters>
      <parameter name="service" type="adcService" prompt="ADC service"/>
    </parameters>
  </persist>

  <actions>
    <action name="configure">
      <devices>
        <device name="adc" device="$service.primary"/>
      </devices>
      <sequence>
        <validateCleanDevice name="check"/>
        <configuration file="myTemplate.vm"/>
        <commit/>
      </sequence>
    </action>
  </actions>
</workflow>
```

In this example, the persistent workflow parameter is an ADC service, which is not an Alteon device but contains a provisioned ADC device that can be retrieved using the `$service.primary` expression. To invoke a configuration template named **myTemplate.vm** on the provisioned ADC of this service, the descriptor defines a local device named "adc" that is created from `$service.primary`. The **validateCleanDevice** and **commit** steps can be used without having to specify which device to operate against because the device has been defined as a local device. Also, because the local device was named "adc," it maps directly to the required parameter name in the template for an ADC device.

An alternative method of creating a dynamic local device connection is to use the **<setDevice />** step to dynamically define or update a device connection. Like persistent and declared local devices, device connections created using **<setDevice />** are supported by the device steps. The difference between a local device connection declared using **<device />** and a dynamic device connection created using **<setDevice />** is that the former is evaluated before the action runs and is guaranteed to connect to the device, while the latter is truly dynamic and may fail to connect to the device.



Note: Always define any Alteon devices accessed as either persistent workflow devices or action local devices. The device workflow steps provide special support for defined devices that are not available if the workflow is unaware that the device is being used.

Sequences

Sequences contain steps. Sequences are used to group a related set of steps. The steps in the sequence run sequentially, one at a time. If any step generates an error, no further steps are executed and the action fails.

[Table 13 - Sequence Attributes, page 113](#) describes the **if** attribute, and the optional **ifState** and **ifNotState** attributes, available for sequences.

Table 13: Sequence Attributes

Attribute	Description
ifState	The value is the name of a state. The entire sequence only runs if the current state of the workflow matches the name specified.
ifNotState	The value is the name of a state. The entire sequence only runs if the current state of the workflow is not the name specified.
If	Any conditional expression that evaluates to true or false. The entire sequence only runs if the expression is true.

In addition to the `<sequence />` element, there is also a `<parallel />` element that behaves just like a sequence except that all of the child steps may run concurrently.

Error Handling

If an error occurs while executing the action, all changes made to the workflow's persistent parameters by the steps in the action before the failure are undone. It is the responsibility of the workflow developer to ensure that any other changes made by the steps are undone using an appropriate error handler. The end result must be the same as if the action had not been run at all. This is required to ensure that workflows are always in a known, reliable state.

The **onError** element is a special kind of sequence that is only executed if an error occurs while executing a step in the action. Multiple error handlers can be specified using the `step` or `afterStep` attributes. All the error handlers for the action that match the specified condition will be executed in the order they are defined. However, only one unqualified error handler may be specified (no `step` or `afterStep` attribute) and it should always be placed last.

[Table 14 - Error Handler Attributes, page 113](#) describes these attributes.

Table 14: Error Handler Attributes

Attribute	Description
step	If provided, the error handler will execute only if the name of the step that failed matches the value of the <code>step</code> attribute.
afterStep	If provided, the error handler will execute only if the step named was executed successfully.

The following shows how to use an error handler in an action:

```
<action name="configure">
  <sequence>
    <validateCleanDevice name="check"/>
    <configuration name="step1" file="configure1.vm"/>
    <configuration name="step2" file="configure2.vm"/>
    <commit/>
  </sequence>
  <onError afterStep="check">
    <autoRevert/>
  </onError>
  <onError step="step1">
    <log message="step1 is the one that failed!"/>
  </onError>
</action>
```

This is a common pattern for error handling. The **validateCleanDevice** step is used to first ensure that the Alteon device has no unapplied or unsaved changes. If this step fails, we do not want any error handler to run. However, if any of the following steps fail, we want to reverse any changes that may have been made to the device. The afterStep="check" error handler does this. It runs only if the error happened after the step named "check" completed successfully.

If and only if the step that failed was named "step1", the error handler with step="step1" will execute and print the message to the audit history.

Parameter Mapping in Steps

Scripting and child workflow steps support parameter mapping. This feature addresses the problem of having differently named parameters in workflows and scripts. When parameters used in a step have the same name as a persistent or local parameter in the workflow, the parameter is automatically made available to the step. But if the two parameters have different names, a mapping is required from the variable used in the workflow to the one used in the step.

The following shows a parameter declaration in a configuration template named **example.vm**:

```
#param($vipAddress, "type=ip", "direction=inout")
```

The following is a workflow that needs to execute this configuration template:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="mappingExample">

  <persist>
    <devices>
      <device name="myDevice"/>
    </devices>
    <parameters>
      <parameter name="vipIp" type="ip" prompt="VIP address"/>
    </parameters>
  </persist>
```

```
(continued)

    <actions>
      <action name="apply">
        <sequence>
          <configuration file="example.vm"/>
        </sequence>
      </action>
    </actions>
  </workflow>
```

The workflow will not work correctly for the following reasons:

- The configuration step will fail because **vipAddress** is a required input parameter for the template, but no variable with that name exists. Instead, in the workflow, the variable is named **vipIp**.
- The configuration template requires a device named "adc", but no device with that name exists in the workflow.

The following shows the set step fixes the problem:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="mappingExample">

  <persist>
    <devices>
      <device name="myDevice"/>
    </devices>
    <parameters>
      <parameter name="vipIp" type="ip" prompt="VIP address"/>
    </parameters>
  </persist>

  <actions>
    <action name="apply">
      <sequence>
        <set saveAs="$vipAddress" value="$vipIp"/>
        <set saveAs="$adc" value="$myDevice"/>
        <configuration file="example.vm"/>
        <set saveAs="$vipIp" value="$vipAddress"/>
      </sequence>
    </action>
  </actions>
</workflow>
```

An action local parameter is created to hold the correctly named parameters needed by the configuration template. Note that because **vipAddress** is an input parameter in the configuration template, it is important to also copy the output value back to the workflow property **vipIp**. This approach is not only verbose and distracting, but it also pollutes the action local parameters with temporary variables that were only needed to invoke a single step.

Using parameter mapping instead solves the problem in a simple, declarative way without creating any additional local parameters:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="mappingExample">

  <persist>
    <devices>
      <device name="myDevice"/>
    </devices>
    <parameters>
      <parameter name="vipIp" type="ip" prompt="VIP address"/>
    </parameters>
  </persist>

  <actions>
    <action name="apply">
      <sequence>
        <configuration file="example.vm">
          <parameterMapping>
            <map from="$vipIp" to="$vipAddress"/>
            <map from="$myDevice" to="$adc"/>
          </parameterMapping>
        </configuration>
      </sequence>
    </action>
  </actions>
</workflow>
```

When using parameter mapping, vDirect recognizes that the **vipAddress** configuration template parameter is both input and output, so it copies the output value back to the **from** variable after the configuration template is finished.

The **from** attribute in the map element always refers to the name or calculated value in the workflow, while the **to** attribute always refers to the name used in the script. The value of the **to** attribute must always be a simple variable name reference, while the value of the **from** attribute can be any expression.

Including Child Workflows

A workflow template can make use of other workflow templates. This is done by creating an instance of another workflow as a child workflow, and storing it as a property. The child workflow can then be updated or deleted by the parent workflow. Child workflows are completely encapsulated within the parent workflow. For example, they cannot be retrieved or updated using the workflow APIs.

Although a child workflow can be created and used as a local property in an action, it is more likely that it will be created and saved as a persistent property of the parent workflow. Because child workflows are not parameters with types and cannot be input parameters, they are defined separately in the persist section of the **workflow.xml** file.

The following shows how to define a persistent child workflow:

```
<persist>
  <parameters>
    ...
  </parameters>
  <workflows>
    <workflow name="configSyncWorkflow"/>
    <workflow name="l3Workflow"/>
  </workflows>
</persist>
```

The configSyncWorkflow property will hold a child workflow and will persist as part of the workflow.

Adding Create and Destroy Actions

By default, the state of a new workflow is "created". Only its persistent properties that have specified default values will be defined. If you want to execute an action as part of the workflow creation, specify the name of the action as the **createAction** attribute of the workflow. vDirect will automatically run this action when creating the workflow.

When an action is used as the **createAction**, its **fromState** is ignored. Because it is specified as the **createAction** for the workflow, vDirect knows that it must be allowed to execute from the "created" state. This allows the developer to reuse **createAction** if required. In addition, any inputs defined for **createAction** become the required inputs to create the workflow.

If the **createAction** fails, the workflow is not created and will not exist in vDirect. Therefore, the workflow developer can guarantee a known, valid state when the workflow is first created. The state of the workflow after it is created will be the **toState** of the **createAction**.

An action can also be set as the **deleteAction** of the workflow. When a workflow is deleted, the specified **deleteAction** is run first. In this case, however, the **fromState** of the **deleteAction** is evaluated so that the **deleteAction** is only run if the current state of the workflow allows it. Otherwise the **deleteAction** is skipped and the workflow is removed from vDirect. When an action is used as the **deleteAction**, its **toState** is ignored and, instead, the final state is set to the predefined "deleted" state. This allows the **deleteAction** to be reused if required. If a **deleteAction** is specified and it fails, the workflow is not deleted.

The create and delete actions are specified as attributes of the workflow element:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="compression"
  createAction="apply"
  deleteAction="remove">
```

The \$workflow Local Variable

Each time an action runs, vDirect injects a variable named \$workflow into the local parameters of the action. This variable provides access to some information about the workflow instance and some utility functions. The \$workflow variable provides the following properties:

- name—The name of this workflow.
- debug —A boolean value that is true if the current action is executing in debug mode.

The `$workflow` variable also provides a connect method that is used to create local device connections in an action. Specifically, the connect method can be used to create local device connections for the provisioned vADCs of an ADC service which is not a device itself and therefore cannot be stored as a device persistent parameter. The following example illustrates how the connect method can be used:

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="localDeviceExample">

  <persist>
    <parameters>
      <parameter name="service" type="adcService" prompt="ADC service"/>
    </parameters>
  </persist>

  <actions>
    <action name="configure">
      <devices>
        <!-- Creates a local connection for the specified device -->
        <device name="adc" device="$workflow.connect($service.primary)"/>
        <!-- Creates a local connection by choosing an
             available adc from the service -->
        <device name="availableAdc" device="$workflow.connect($service)"/>
      </devices>
      <sequence>
        <!-- Add steps here ->
        <!-- commit now works as expected with the local devices -->
        <commit/>
      </sequence>
    </action>
  </actions>
</workflow>
```

Device Variables

Any variable that is a device connection, regardless of how it was obtained (from a `<device />` declaration or `<setDevice />` step, is an object with the following methods that can be used in the template. These variables are analogous to the device variables created by **#device** directives in configuration templates described at [Table 15 - Device Variables, page 119](#).

Table 15: Device Variables

Method	Description
Object newBean(String beanClassOrCLIPath)	Creates an empty instance of one of the vDirect Device SDK JavaBeans supported by the device. The name supplied can be either the simple class name of the bean to create, or it can be a CLI command path if supported by the device. For more information, see Identifying Bean Classes, page 66 . The object returned is an instance of the vDirect bean, and the bean methods can then be called for it.
Object readBean(String beanClassOrCLIPath)	Same as Object newBean, but it reads the bean from the device and the returned object with all the values defined.
Object read(Object bean)	Reads the bean object specified from the device and returns an object of the same type with all the values defined.
List<?> readAllBeans(String beanClassOrCLIPath)	Reads all the entries in a table from the device and returns them as a list.
Object findFirst(Object bean)	Finds the first object on the device that matches the filled in fields of the bean specified.
List<?> findAll(Object bean)	Finds all the objects on the device that match the filled in fields of the bean specified.
Object getFirstFreeIndex(String beanClassOrCLIPath)	Finds the first free index in the specified table.
Object getFreeIndex(Object bean)	Finds the first free index in a table after the bean specified.
Object getFreeIndex(String beanClassOrCLIPath, Object startValue)	Finds the first free index in the specified table after the start value.
Object getFreeIndexWithDefault(Object bean)	Same as getFreeIndex, but it returns the bean specified if its index is free.
Object getFreeIndexWithDefault(String beanClassOrCLIPath, Object startValue)	Same as getFreeIndex, but it uses the specified start value if it is available in the table.
boolean exists(Object bean)	Returns true if the specified table entry exists on the device; false otherwise.
boolean isEmpty(Object bean)	Returns true if the specified bean is null or empty.
boolean isEmpty(Object bean)	Returns true if the specified bean is not null and not empty
boolean isNull(Object bean)	Convenience method to test if a returned bean object is null.
boolean isNotNull(Object bean)	Convenience method to test if a returned bean object is not null.
boolean setEnumValue(Object bean, String property, String value)	Sets the value of a JavaBean property when the property is a Java enum type. This method allows the template to simply use the string name of the desired enum value.

Table 15: Device Variables (cont.)

Method	Description
Iterator iterate(String beanClassOrPath)	Returns a table iterator that can be used with the Velocity <code>#foreach</code> directive.
boolean isAlteonMaster()	Returns true if the device is an Alteon, is reachable, and its HA state is NONE, MASTER, or ACTIVE.
boolean isNetworkReachable(int timeout)	Returns true if the device is network reachable within the specified timeout in milliseconds. Reachable means ICMP ECHO or some equivalent test.
boolean testConfigProtocol(timeout)	Returns true if the configuration protocol specified for the device is working. This means the device is responding and the credentials vDirect is using are correct. The protocol is either HTTP or SNMP depending on the registered device configuration. The timeout is in milliseconds.
String getDeviceName()	Returns a name of the registered device from the vDirect inventory. If there is no backing registered device, the host name is returned.
void update (Object bean)	Updates the object on the device. The object passed must be one of the vDirect factory beans.
void delete (Object bean)	Removes a table entry from the device. The object passed must be one of the vDirect beans representing a table entry.
String downloadAlteonConfiguration(String password)	Gets an Alteon configuration dump. If the password is not null the dump will include Alteon private keys encrypted using the password.
void uploadAlteonConfiguration(String configuration, String password)	Uploads the Alteon configuration dump. If the password is not null it will be used to restore any private keys found in the configuration.

DefensePro Devices

The following Device Adapter methods can be used when the device is a DefensePro.

Table 16: DefensePro Device Variables

Method	Description
ExportServerPolicyOptions getExportServerPolicyOptions ()	Use to configure options for exporting DefensePro server policies.
ImportExportResult exportServerPolicy (String policyName)	Exports the named policy from the DefensePro.
ExportNetworkPolicyOptions getExportNetworkPolicyOptions ()	Use to configure options for exporting DefensePro network policies.
ImportExportResult exportNetworkPolicy (String policyName)	Exports the named policy from the DefensePro.
ExportSubdomainsWhitelistOptions getExportSubdomainsWhitelistOptions()	Use to configure options for exporting DefensePro sub-domain whitelists.

Table 16: DefensePro Device Variables (cont.)

Method	Description
ImportExportResult exportSubdomainsWhitelist (String policyName, String dnsProfileName)	Exports the whitelist for the named policy from the DefensePro.
ImportPolicyOptions getImportPolicyOptions ()	Use to configure common import options for server and network policies.
ImportExportResult importNetworkPolicy (String policy)	Imports the previously exported policy to the DefensePro.
ImportExportResult importServerPolicy (String policy)	Imports the previously exported policy to the DefensePro.
ImportSubdomainsWhitelistOptions getImportSubdomainsWhitelistOptions()	Use to configure options for importing a sub-domains whitelist.
ImportExportResult importSubdomainsWhitelist (String policyName, String dnsProfileName, String whitelist)	Imports a previously exported whitelist for the specified policy and DNS profile.

Workflow Steps

The following sections document the types of steps that can be used in action sequences. All steps have an optional name and if attribute.

Table 17: Workflow Step Attributes

Attribute	Description
name	Optional name for the step that will be displayed in history messages and can also be referred to in error handlers.
if	Optional conditional expression. The step is executed only if the expression is true.

Device Steps

Device steps operate against the Alteon devices in the scope of the action. By default these steps will perform their function against all of the local devices, one at a time. However, the template author can use the optional devices element to override this default behavior and explicitly define which devices are to be acted against.

The following workflow.xml snippet shows how the devices element can be used to perform a revert only on the device named "adc1:"

```
<action name="setup">
  <sequence>
    <configuration name="step1" file="setup.vm"/>
    <revert name="step2">
      <devices>
        <device name="adc1"/>
      </devices>
    </revert>
  </sequence>
</action>
```

device

This is a declarative step used both with persistent and action input parameters, as well as for declaring device local connections. It has the following attributes:

Table 18: device Attributes

Attribute	Description
name	The name of the variable that will hold the device connection
type	The type of the device. Values : alteon, defensePro, appWall. Default: alteon
minLength	If the variable is an array of devices this value is the required minimum number of devices in the array.
maxLength	If the variable is an array of devices this value is the maximum number of devices allowed for the array. The value -1 is used to indicate no maximum.
driverVersion	The version of the device driver that vDirect should use, regardless of the actual version of the device. Optional.
deviceVersion	The actual version of the device. Mandatory. Note: vDirect will interrogate the device before executing the template to ensure that this requirement is met.
device	This is an expression whose value is the actual device connection. This attribute can only be used in an action local device declaration.
if	An optional boolean expression. If the expression evaluates false, then no device connection is created. This attribute can only be used in an action local device declaration.

setDevice

The **setDevice** step dynamically defines a device connection during the execution of the workflow. Under some circumstances it can be more flexible to use **<setDevice />** than static **<device />** declarations. **<setDevice />** is similar to the **<set />** step, except that vDirect will know that the variable is a device connection and all of the other device steps will be supported for any device defined using **<setDevice />**. It has the following attributes:

Table 19: setDevice Attributes

Attribute	Description
saveAs	The context variable to assign the device connection to.
value	An expression that must evaluate to a device connection, or an array of device connections.

In the following example, an action takes only an ADC service as input and defines no local devices at all. It creates two device connections dynamically, **\$primary** and **\$secondary**, both of which are local to the **<sequence/>** block they are defined in. A third device connection named **\$master** is assigned to one of these values and is accessible outside the **<parallel/>** block.

<setDevice/> allows the workflow to dynamically create or alter existing device connections.

```
<action name="testSetDevice">
  <inputs>
    <parameters>
      <parameter name="service" type="adcService" prompt="service"/>
      <parameter name="serverGroup" type="string" prompt="Server group name"/>
    </parameters>
  </inputs>

  <sequence>
    <setDevice saveAs="$master"/>
    <parallel>
      <sequence>
        <setDevice saveAs="$primary"
          value="$workflow.connect($service.primary)"/>
        <log message="Primary ADC is $primary"/>
        <if>
          <and>
            <isTrue value="$primary.testConfigProtocol(5000)"/>
            <isTrue value="$primary.alteonMaster"/>
          </and>
          <then>
            <log message = "Primary is available and master"/>
            <setDevice saveAs="$master" value="$primary"/>
          </then>
          <else>
            <log message = "Primary is not available or not master"/>
          </else>
        </if>
      </sequence>
    </parallel>
  </sequence>
</action>
```

```
(continued)

</if>
</sequence>

<sequence>
  <setDevice saveAs="$secondary"
    value="$workflow.connect($service.secondary)"/>
  <log message="Secondary ADC is $secondary"/>
  <if>
    <and>
      <isTrue value="$secondary.testConfigProtocol(5000)"/>
      <isTrue value="$secondary.alteonMaster"/>
    </and>
    <then>
      <log message = "Secondary is available and master"/>
      <setDevice saveAs="$master" value="$secondary"/>
    </then>
    <else>
      <log message = "Secondary is not available or not master"/>
    </else>
  </if>
</sequence>
</parallel>
<if>
  <isTrue value="$master"/>
  <then>
    <configuration file="get-servers.vm" name="master">
      <parameterMapping>
        <map from="$master" to="$adc"/>
      </parameterMapping>
    </configuration>
    <apply device="$master"/>
  </then>
  <else>
    <error message="There is no master!"/>
  </else>
</if>
</sequence>
</action>
```

validateCleanDevice

This step validates that each device has no unapplied changes and no unsaved configuration. This is often used as the first step in a sequence to ensure that Alteon devices are in a known state that can be recovered in case of an error. If any device has unapplied changes or unsaved configuration, the action will fail. It has the following attributes:

Table 20: validateCleanDevice Attributes

Attribute	Description
unapplied	This is an optional boolean expression. If the expression evaluates to true, then if any device has unapplied changes the validate step will fail. The default is true.

Table 20: validateCleanDevice Attributes (cont.)

Attribute	Description
unsaved	This is an optional Boolean expression. If the expression evaluates to true, then if any device has unsaved changes the validate step will fail. The default is true.

apply

This step will apply the new configuration to each Alteon. AlteonIf any device fails to apply successfully, the action will fail.

save

This step will save the configuration on each Alteon. If any device fails to save successfully, the action will fail.

revert

This step will revert the new unapplied configuration on each Alteon. If any Alteon fails to revert successfully, the action will fail.

revertApply

This step will revert to the current unsaved configuration on each Alteon. AlteonIf any device fails to revert apply successfully, the action will fail.

commit

This step will perform a revert on each Alteon, and then perform a save on each Alteon. The commit step is often used with autoRevert. Alteon

autoRevert

This step acts like a revert step for each Alteon that has unapplied configuration, or it acts like a revertApply step for each device that apply successfully completed. The autoRevert step works with the other device steps and is most useful in an error handler. In your action sequence you can use the apply, save, or commit steps and then in case of error simply use a single autoRevert step to remove the changes made by the workflow. Note that autoRevert may not work correctly if you perform an Alteon apply or save operation from inside a configuration template or groovy script. For this reason, it is recommended that you use the device steps to perform these actions on Alteon devices. Alteon

Child Workflow Steps

Workflow steps that support child workflows allow you to create, update, and delete child workflows. Each of the workflow steps supports parameter mapping.

createWorkflow

This step creates a new instance of a workflow from a template and stores the new workflow instance as a property. It has the following attributes:

Table 21: createWorkflow Attributes

Attribute	Description
workflow	A variable name used to store the workflow.

Table 21: createWorkflow Attributes

Attribute	Description
template	The name of the workflow template to create.

The template must be previously installed in vDirect. Also note that any inputs required to create the child workflow must be supplied either by the local properties or by using parameter mapping. The object created by the createWorkflow step supports a special notation for accessing the persistent properties of the child workflow. The persistent properties of the child workflow are accessible directly as indexed properties in Velocity expressions, as the following example shows:

```
<sequence>
  <createWorkflow workflow="$myChildWorkflow" template="compression"/>
  <set saveAs="$defaultLevel" value="{myChildWorkflow['complvl']}" />
</sequence>
```

updateWorkflow

This step updates an existing child workflow by invoking an action as defined by the child workflow's template. Note that any inputs required to create the child workflow must be supplied either by the local properties or by using parameter mapping.

Table 22: updateWorkflow Attributes

Attribute	Description
workflow	An expression that refers to the child workflow.
action	An expression that evaluates to the name of the action to execute.

deleteWorkflow

This step deletes an existing child workflow, executing any delete actions defined by the workflow.

Table 23: updateWorkflow Attribute

Attribute	Description
workflow	An expression that refers to the child workflow to delete.

Scripting Steps

The following two scripting steps perform the bulk of the work in any workflow:

- [configuration, page 127](#)
- [script, page 127](#)

The scripting steps support parameter mapping. The configuration templates and groovy scripts that are used by the workflow are included as separate files in the workflow archive, and they are referenced using the file attribute of the scripting step.

Table 24: Scripting Step Attribute

Attribute	Description
file	The file name of the script or configuration template to execute in the ZIP archive.

In addition, the scripting steps support the ability to specify which of the action's local properties are visible to the script, both as inputs and outputs using the inputs and outputs elements. This is used to override the default behavior that all local properties are visible to the script, and that all outputs from the script are copied back into the local properties.

Table 25: Scripting Elements

Attribute	Description
inputs	Restricts the action's local properties that are visible to the step to only those specified as parameter elements in the inputs element.
outputs	Restricts the outputs from the step that are copied back into the action's local properties to only those specified as parameter elements in the outputs element.

configuration

This step runs a vDirect configuration template. Unlike groovy scripts, configuration templates must declare their inputs and outputs using the `#param` directive. vDirect workflows take advantage of this to automatically configure the inputs- only the local properties of the action that have the same name as a configuration template input parameter will be passed into the configuration template. The workflow template author must use parameter mapping to map any required inputs that have a different name than that used in the workflow or that are calculated from other workflow properties.

Likewise, the workflow engine automatically copies only the defined configuration template output parameters back into the local properties of the action. Again, parameter mapping can be used to map the configuration template outputs to a complex expression in the workflow's properties.

Configuration templates do not have to be located in the workflow archive. Instead of using the `file` attribute to refer to the configuration template, the workflow can execute any installed global configuration template by instead using the `ref` attribute:

Table 26: Configuration Attribute

Attribute	Description
ref	Use instead of <code>file</code> to refer to the name of a global configuration template previously installed in vDirect.



Note: Configuration templates that are included in a workflow archive are visible only to the workflow. They are not visible as global configuration templates installed in vDirect.

script

This step runs a groovy script. By default, the groovy script has access to all of the action's local properties for both input and output. These properties are automatically defined as variables in the script binding and can be used directly. Use the inputs and outputs elements to override this behavior.

In addition to any input parameters provided by the workflow, two additional variables are added to the script binding:

Table 27: Scripting Elements

Variable	Description
log	An instance of org.slf4j.Logger that can be used to log messages.
vdirect	An instance of com.radware.vdirect.server.VDirectServerClient that can be used to call any vDirect APIs.

If the script returns a result, the calling workflow can access it as the value of a parameter named "result".

Resource Steps

The resource steps simplify allocating resources from vDirect resource pools. The resource pools that can be used include IP address pools, VRRP pools, and ISL pools.

This section includes the following resource steps:

- [acquireResource, page 128](#)
- [releaseResource, page 128](#)
- [autoReleaseResource, page 129](#)

acquireResource

This step allocates resource from a resource pool and saves the acquired resource in a local property. The local property will almost always be a persistent parameter of the workflow so that it can be released later when the workflow is deleted.

Table 28: acquireResource Attributes

Attribute	Description
pool	An expression that evaluates to a vDirect resource pool.
saveAs	A variable name that will be used to store the acquired resource.
owner	An optional expression that will be recorded as the owner of the acquired resource. The default value is the name of this workflow instance.
comment	A required expression that will be recorded as the comment field for the resource acquisition.

releaseResource

This step releases an acquired resource back to the resource pool it was allocated from.

Table 29: releaseResource Attributes

Attribute	Description
pool	An expression that evaluates to a vDirect resource pool.
resource	An expression that evaluates to the resource to release (e.g. an IP address).

autoReleaseResource

This step can be used as part of an error handler to automatically release any resources that were acquired by the acquireResource step in the current action. Note that autoReleaseResource cannot release resources that may have been acquired using groovy scripts or configuration templates and therefore the acquireResource step should be used for this purpose. The autoReleaseResource step has no attributes.

Conditional Execution Steps

vDirect supports conditional execution of workflow steps using the **<if />** element.



Note: The **<if />** element can only be used inside an outer statement block (a **<sequence />** or **<parallel />** block). For example, it cannot be used inside the **<inputs />** element of an action.

The following example illustrates how to use the **<if />** element:

```

<if>
  <equals arg1="${foo}" arg2="bar" />
  <then>
    <log message="The value of property foo is bar" />
  </then>
  <else>
    <log message="The value of property foo is not bar" />
  </else>
</if>

<if>
  <equals arg1="${foo}" arg2="bar" />
  <then>
    <log message="The value of property foo is 'bar'" />
  </then>

  <elseif>
    <equals arg1="${foo}" arg2="foo" />
    <then>
      <log message="The value of property foo is 'foo'" />
    </then>
  </elseif>

  <else>
    <log message="The value of property foo is not 'foo' or 'bar'" />
  </else>
</if>

```

The formal grammar for the **<if />** element is:

IF := "<if>" CONDITION THEN ELSEIF* ELSE? "</if>"

THEN and ELSE are just plain elements with no attributes:

THEN := "<then>" STATEMENTS "</then>"

ELSE := "<else>" STATEMENTS "</else>"

ELSEIF requires a condition as the first child:

ELSEIF := "<elseif>" CONDITION THEN "</elseif>"

The following are the supported conditions:

CONDITION := AND | OR | EQUALS | NOT | ISTRUE | ISFALSE | MATCHES | ISEMPY

AND := "<and>" CONDITION 2+ "</and>"

OR := "<or>" CONDITION 2+ "</or>"

Both AND and OR require at least two child condition elements to be valid. The AND and OR steps are short-circuiting so that only some of the child conditions may be evaluated.

The remaining conditions are:

NOT := "<not>" CONDITION "</not>"

EQUALS := "<equals arg1="expression" arg2="expression"/>"

ISTRUE := "<isTrue value="expression"/>"

ISFALSE := "<isFalse value="expression"/>"

MATCHES := "<matches string="expression" pattern="regular_expression" caseSensitive? multiline? dotall? />"

ISEMPY := "<isEmpty value="expression"/>"

ISEMPY returns true if the value is null, or is an empty string or collection, or is an empty vDirect factory bean.

Utility Steps

This section includes the following utility steps:

- [log, page 130](#)
- [set, page 130](#)
- [error, page 131](#)
- [forEach, page 131](#)

log

This step creates an entry in the workflow audit history.

Table 30: Log Attributes

Attribute	Description
level	Optional log level. The default is info. The allowed values are debug, info, warn, and error.
message	An expression that will be rendered as the log message.

set

This step can be used to set the value of any property in the action's local parameters, or to create a new local parameter.

Table 31: Set Attributes

Attribute	Description
saveAs	A variable expression to set.
value	An expression that evaluates to the value to set.

error

This step can be used to generate an error and halt execution of the current action. The message is logged in the audit history as an error.

Table 32: Error Attributes

Attribute	Description
message	An expression that evaluates to the text of the log message stored in the workflow's audit history.

forEach

This step can be used to iterate over a workflow. The `forEach` step includes optional `name` and `in` attributes in the format `<forEach var="{name}" in="{expression}"/>`. For example:

```
<sequence>
  <forEach var="$i" in="[1,2,3]">
    <forEach var="$j" in="[1,2,3]">
      <log message="inner j = $j forEach.index = $forEach.index"/>
    </forEach>
    <log message="outer i = $i forEach.index = $forEach.index"/>
  </forEach>
</sequence>
```

The `forEach` has an optional attribute `parallel=true`. When set, each iteration of the loop runs concurrently and receives its own local copy of the variable binding.

CHAPTER 8 – INPUT AND OUTPUT PARAMETER TYPES

Both configuration templates and workflows can define input and output parameters using **#param** and `<parameter />` respectively.

The supported parameter types are the same in both cases and are described in [Table 33 - Parameter Types, page 133](#).



Note: Some of the parameter types do not make sense in configuration templates and are expected to be used only in workflow templates.

Parameter Type Reference

For information on using the **#param** directive in configuration templates, see [Parameter Directive, page 93](#).

For information on using the `<parameter/>` element in the descriptor for workflow templates, see [Workflow Descriptor, page 105](#).

Table 33: Parameter Types

Type	Description
adcService	The parameter is a reference to an ADC service in the inventory.
addressPool	The parameter is a reference to an IP address pool in the inventory.
bool	The parameter is a boolean value.
bytea	The parameter value is an array of bytes. The template or workflow will receive the value as a Java byte[]. The vDirect Web user interface will prompt for a bytea input parameter using a file selection dialog. To programmatically send a bytea input parameter value in a JSON payload, the value must be sent as a BASE64 encoded string.
containerResourcePool	The parameter is a reference to a container resource pool.
credentials	The parameter value holds device credentials. In this case, the format attribute must be specified and must be one of <code>userNamePassword</code> , <code>communityString</code> , or <code>snmpv3</code> . The parameter cannot be an output parameter.
deviceCollection	The parameter is a reference to a collection of devices.
double	The parameter is a decimal value.
int	The parameter is an integer value.
ip	The parameter is an IPv4 or IPv6 address.
ipv4	The parameter is an IPv4 address.
ipv6	The parameter is an IPV6 address.
islId	The parameter is an Alteon ISL ID.

Table 33: Parameter Types

Type	Description
islPool	The parameter is a reference to an Alteon ISL pool in the inventory.
long	The parameter is a long integer value.
network	The parameter is a reference to a Network in the inventory.
object	The parameter is an opaque object passed as a hashmap. vDirect does not perform any validation in this case.
string	The parameter is a string value.
vrrpPool	The parameter is a reference to a VRRP pool in the inventory.



Notes

- Whenever the type of a parameter is unspecified, a string parameter is used by default.
- If a parameter type that is not listed in [Table 33 - Parameter Types, page 133](#) is used, it must have the name of a user-defined type.
- Any parameter can be defined as an array by appending [] to the type. For example, `string[]` means the parameter is an array of strings. For array parameters, use the `minlength` and `maxlength` attributes to constrain the allowed size of the array.
- For a credentials parameter, the end user can select an existing entry from the credentials store, or enter credentials directly. This may allow end users to execute a script against a device without knowing the device credentials. Permissions can be used to control this behavior.

The parameter types that are input using an `AdcResourceId` are wrapped using special objects that make them easy to work with inside configuration templates or workflow expressions.

Various methods can be called on these wrapped objects:

- Wrapped resource pools—The template types **addressPool**, **islPool**, and **vrrpPool** are wrapped using objects that have the following methods to acquire and release resources from the pool:

Table 34: Wrapped Resource Pool Methods

Method	Description
<code>acquire (String owner, String comment)</code>	Acquires a resource from the pool.
<code>acquire (Object resource, String owner, String comment)</code>	Tries to acquire the specified resource from the pool.
<code>release ()</code>	Releases a specific resource from the pool.
<code>revert ()</code>	Releases all resource acquired using <code>acquire()</code> .
<code>getAcquired()</code>	Returns an array of the resources acquired by this instance of the wrapped pool.
<code>getId()</code>	Returns the <code>AdcResourceId</code> of the resource pool that is wrapped.
<code>getResourcePool()</code>	Returns the underlying vDirect SDK resource pool that is wrapped.

- The template type `network` is wrapped using an object with the following methods:

Table 35: Wrapped Template Type Network Methods

Method	Description
getAddressPool(String name)	Returns the wrapped IP address pool for the specified name.
getId()	Returns the AdcResourceId of this network.
getNetwork()	Returns the underlying vDirect SDK Network object that is wrapped.
getVlan()	Returns the integer VLAN assigned to this network.
getVrrpPool()	Returns the wrapped VRRP resource pool assigned to this network.

- The template type **containerResourcePool** is wrapped using an object with the following methods:

Table 36: Wrapped Template Type containerResourcePool Methods

Method	Description
getId()	Returns the AdcResourceId of this container resource pool.
getIslPool()	Returns the wrapped ISL resource pool assigned to this container resource pool.
getManagementAddressPool()	Returns the wrapped management IP address pool assigned to this container resource pool.
getNetwork(String name)	Returns the wrapped network for the specified name.
getResourcePool()	Returns the underlying vDirect SDK AdcResourcePool object that is wrapped.
getVlanNetwork(int vlan)	Returns a wrapped network for the specified VLAN

- The template type **adcService** is wrapped using an object with the following methods:

Table 37: Wrapped Template Type adcService Methods

Method	Description
getId()	Returns the AdcResourceId of this ADC service.
getIslId()	Returns the provisioned IslId object for this ADC service.
getRequest()	Returns the vDirect SDK AdcServiceRequest object used too provision this service.
getPrimary(boolean provision)	Returns an AdcCLIConnection object for the provisioned primary ADC. If the service is concurrently provisioning, this call will wait for provisioning to complete before returning. If the provision argument is true and the service is not provisioned, vDirect attempts to provision the service.
getPrimaryId(boolean provision)	Returns the AdcResourceId of the provisioned primary ADC. If the provision argument is true and the service is not provisioned, vDirect attempts to provision the service.
getSecondary(boolean provision)	Returns an AdcCLIConnection object for the provisioned secondary ADC, if any. If the provision argument is true and the service is not provisioned, vDirect attempts to provision the service.

Table 37: Wrapped Template Type adcService Methods

Method	Description
getSecondaryId(boolean provision)	Returns the AdcResourceId of the provisioned secondary ADC. If the provision argument is true and the service is not provisioned, vDirect attempts to provision the service.
getService()	Returns the underlying vDirect SDK AdcService object that is wrapped.
getState()	Returns the current state of the ADC service.

CHAPTER 9 – AUTHORIZING WIZARDS

You can use workflows to implement a wizard. This section describes how to use the following features to author a wizard:

- [autoName, page 137](#)
- [RunAs Sequence, page 137](#)
- [<runNext> Element, page 138](#)
- [Sequence Termination, page 139](#)
- [Navigating Back, page 141](#)

autoName

If the entire workflow lifecycle is only for the purpose of a wizard, developers should use the **autoName** feature to prevent users from having to supply a name for the workflow. In this case, the initial action is a **createWorkflow** action on the workflow template, while the following actions would be on the workflow instance and end with the automatic deletion of the workflow.

Note that **autoName** can be used regardless of whether or not the workflow implements a wizard. It is a separate feature.

```
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="alteon-wizard-example"
  createAction="init"
  autoName="true">
```

When the **autoName** attribute is set to true, the name of the workflow becomes an optional input parameter to the create action. The **workflowName** parameter will not appear in the list of required inputs returned by the runnable APIs. If a name is not provided, vDirect will generate a unique name for the workflow.

RunAs Sequence

Actions have a new attribute **runAs** which offers a hint to clients about how to run the action. When the **runAs** attribute is set to **sequence**, the action promises to return a **RunNextResult** object as the result if the action successfully completes. In other words, the result of running the action is an instruction about the next action to run.

```
<action name="init" visible="false" runAs="sequence">
  <inputs>
    <devices>
      <device name="adc"/>
    </devices>
  </inputs>
</sequence>
  <set saveAs="$groups" value="$adc.readAllBeans('enh/cfg/slb/group')"/>
  <error if="$groups.empty" message="There are no server groups on this
device!"/>
```

```
(continued)

<set saveAs="$groupNames" value="[]" />
  <forEach var="$group" in="$groups">
    <set saveAs="$dummy" value="$groupNames.add($group.index)" />
  </forEach>
</sequence>

<runNext action="chooseGroup" cancelAction="deleteWorkflow">
  <parameters>
    <parameter name="group" values="$groupNames" />
  </parameters>
</runNext>
</action>
```



Note: When a **runAs** sequence is specified, the workflow *must* include the special **<runNext/>** element in place of the normal **<result/>** element.

<runNext> Element

This element instructs the caller about the next action in the workflow to run. In addition, the **<runNext/>** element can redefine attributes of the input parameters and also define hidden arguments that the caller should pass to the next action.

The **<runNext/>** element has the following attributes:

Table 38: <runNext> Attributes

Attribute	Description
action	The required name of the action in the workflow to run next.
cancelAction	<p>An optional action that the client <i>should</i> run if not calling the next action or if the next action fails.</p> <p>Use this action to delete an autoName workflow that is used to implement the wizard.</p> <p>This attribute has the following nested elements:</p> <ul style="list-style-type: none"> parameters—The same as the parameters element used to define inputs to an action. The runNext element can use this to modify the input parameters that the client sees for the next action. <p>Note: This cannot be used to create or remove any parameters from the next action, it can only be used to modify attributes of those parameters.</p> <ul style="list-style-type: none"> args—A set of name-value pairs that the client must pass unmodified as inputs to the next action. If the name of an argument is the same as an input parameter for the next action, that parameter will be removed the list of required inputs to the next action. In this case the end-user will never see the affected parameter.

Both **action** and **cancelAction** attributes can be expressions that refer to a variable in the local context. There is no need for a conditional statement to assign the action in the **runNext** element because the action sequence can setup any required variables that can then be referred to in the **action** attribute.

Sequence Termination

An action with **runAs = sequence** can be followed by another action that is also run as a sequence and so on until finally the last action is not a sequence. This action corresponds to the final result of the wizard. If an **autoName** workflow is used, this action should also delete the workflow.

Example

The following workflow example takes an Alteon device as input at creation and stores it as a persistent parameter (where it can be accessed by any action in the sequence). The create action reads the available server groups from the Alteon and modifies the required input parameter to the next action group so that the user can only select from an available group.

Once the group is chosen, it is passed back to the client in the **runNext args** so that the client will then pass it to the final action, **showServers**. An alternative would be to also store the selected group as a persistent parameter—either way can work.

The final action **showServers** sets the final state to be deleted, which automatically causes the workflow to be deleted after the action completes.

```
<?xml version="1.0"?>
<workflow
  xmlns="http://www.radware.com/vdirect"
  name="alteon-wizard-example"
  createAction="init"
  autoName="true">

  <description>Transient wizard example</description>

  <persist>
    <devices>
      <device name="adc" type="alteon"/>
    </devices>
  </persist>
<actions>

  <action name="init" visible="false" runAs="sequence">
    <inputs>
      <devices>
        <device name="adc"/>
      </devices>
    </inputs>
    <sequence>
      <set saveAs="$groups" value="$adc.readAllBeans('enh/cfg/slb/group')"/>
      <error if="$groups.empty" message="There are no server groups on this
device!"/>
    </sequence>
    <set saveAs="$groupNames" value="[]"/>
    <forEach var="$group" in="$groups">
      <set saveAs="$dummy" value="$groupNames.add($group.index)"/>
    </forEach>
  </action>

  <runNext action="chooseGroup" cancelAction="deleteWorkflow">
```

```
(continued)

<parameters>
    <parameter name="group" values="$groupNames"/>
</parameters>
</runNext>
</action>

<action name="chooseGroup" visible="false" runAs="sequence">
    <inputs>
        <parameters>
            <parameter name="group" type="string" prompt="Choose real server
group"/>
        </parameters>
    </inputs>
    <sequence>
        <log message="Selected group is $group"/>
    </sequence>

    <runNext action="showServers" cancelAction="deleteWorkflow">
        <args>
            <arg name="group" value="$group"/>
        </args>
    </runNext>
</action>
<action name="showServers" toState="deleted" visible="false">
    <inputs>
        <parameters>
            <parameter name="group" type="string" prompt="Choose real server
group"/>
        </parameters>
    </inputs>
    <outputs>
        <parameter name="servers"/>
    </outputs>
    <sequence>
        <set saveAs="$servers" value="[]"/>
        <forEach var="$entry"
in="$adc.readAllBeans('SlbNewCfgEnhGroupRealServerEntry', $group)">
            <set saveAs="$server"
value="$adc.readBean('SlbNewCfgEnhRealServerEntry', $entry.ServIndex)"/>
            <set saveAs="$dummy" value="$servers.add($server)"/>
        </forEach>
        <set saveAs="$ip" value="$adc.host"/>
    </sequence>
    <result type="text/html" template="html-servers.vm"/>
</action>
</actions>
</workflow>
```

Navigating Back

Workflow developers creating wizards by using sequence actions should understand that users may interactively navigate back through previous pages of the wizard and then forward again, each time repeating the sequence actions associated with each page.

As a general rule it is strongly recommended that no sequence action should have any mutating effects until the last action in the sequence is reached.

This simply policy avoids any unexpected consequences from repeating an action in the sequence. If the workflow developer needs to know that the user went back to a previous page in the wizard, they may set a cancel action in the **RunNextResult**. The vDirect Web UI ensures that this action is called when the Back button is pressed.

Any workflow **create** action that is part of a sequence must specify a **backAction** of **deleteWorkflow** to prevent creating orphaned workflow instances each time the user advances forward from the first page of any such wizard.

For backward compatibility with existing workflows written prior to this feature, vDirect automatically sets the **backAction** in this case to **deleteWorkflow** if one is not already set.

When the vDirect Web UI calls the back action, it will supply no inputs and need not wait for completion. A back action takes no inputs and return no result or outputs.

The **backAction** is set as an attribute using the name of the action to run:

```
<action name="init" visible="false" runAs="sequence">
  <inputs>
    <parameters>
      <parameter name="x"/>
    </parameters>
  </inputs>
  <runNext action="page2" cancelAction="deleteWorkflow"
backAction="deleteWorkflow">
    </runNext>
  </action>
```

The **<runNext/>** element has a boolean **showBackButton** attribute that can be set to false to prevent the user from going back to the previous step in a wizard.

```
<runNext action="page2" cancelAction="deleteWorkflow" showBackButton="false">
</runNext>
```


RADWARE LTD. END USER LICENSE AGREEMENT

By accepting this End User License Agreement (this "License Agreement") you agree to be contacted by Radware Ltd.'s ("Radware") sales personnel.

If you would like to receive license rights different from the rights granted below or if you wish to acquire warranty or support services beyond the scope provided herein (if any), please contact Radware's sales team.

THIS LICENSE AGREEMENT GOVERNS YOUR USE OF ANY SOFTWARE DEVELOPED AND/OR DISTRIBUTED BY RADWARE AND ANY UPGRADES, MODIFIED VERSIONS, UPDATES, ADDITIONS, AND COPIES OF THE SOFTWARE FURNISHED TO YOU DURING THE TERM OF THE LICENSE GRANTED HEREIN (THE "SOFTWARE"). THIS LICENSE AGREEMENT APPLIES REGARDLESS OF WHETHER THE SOFTWARE IS DELIVERED TO YOU AS AN EMBEDDED COMPONENT OF A RADWARE PRODUCT ("PRODUCT"), OR WHETHER IT IS DELIVERED AS A STANDALONE SOFTWARE PRODUCT. FOR THE AVOIDANCE OF DOUBT IT IS HEREBY CLARIFIED THAT THIS LICENSE AGREEMENT APPLIES TO PLUG-INS, CONNECTORS, EXTENSIONS AND SIMILAR SOFTWARE COMPONENTS DEVELOPED BY RADWARE THAT CONNECT OR INTEGRATE A RADWARE PRODUCT WITH THE PRODUCT OF A THIRD PARTY (COLLECTIVELY, "CONNECTORS") FOR PROVISIONING, DECOMMISSIONING, MANAGING, CONFIGURING OR MONITORING RADWARE PRODUCTS. THE APPLICABILITY OF THIS LICENSE AGREEMENT TO CONNECTORS IS REGARDLESS OF WHETHER SUCH CONNECTORS ARE DISTRIBUTED TO YOU BY RADWARE OR BY A THIRD PARTY PRODUCT VENDOR. IN CASE A CONNECTOR IS DISTRIBUTED TO YOU BY A THIRD PARTY PRODUCT VENDOR PURSUANT TO THE TERMS OF AN AGREEMENT BETWEEN YOU AND THE THIRD PARTY PRODUCT VENDOR, THEN, AS BETWEEN RADWARE AND YOURSELF, TO THE EXTENT THERE IS ANY DISCREPANCY OR INCONSISTENCY BETWEEN THE TERMS OF THIS LICENSE AGREEMENT AND THE TERMS OF THE AGREEMENT BETWEEN YOU AND THE THIRD PARTY PRODUCT VENDOR, THE TERMS OF THIS LICENSE AGREEMENT WILL GOVERN AND PREVAIL. PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THE PACKAGE CONTAINING RADWARE'S PRODUCT, OR BEFORE DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING RADWARE'S STANDALONE SOFTWARE (AS APPLICABLE). THE SOFTWARE IS LICENSED (NOT SOLD). BY OPENING THE PACKAGE CONTAINING RADWARE'S PRODUCT, OR BY DOWNLOADING, INSTALLING, COPYING OR USING THE SOFTWARE (AS APPLICABLE), YOU CONFIRM THAT YOU HAVE READ AND UNDERSTAND THIS LICENSE AGREEMENT AND YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT. FURTHERMORE, YOU HEREBY WAIVE ANY CLAIM OR RIGHT THAT YOU MAY HAVE TO ASSERT THAT YOUR ACCEPTANCE AS STATED HEREINABOVE IS NOT THE EQUIVALENT OF, OR DEEMED AS, A VALID SIGNATURE TO THIS LICENSE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT, YOU SHOULD PROMPTLY RETURN THE UNOPENED PRODUCT PACKAGE OR YOU SHOULD NOT DOWNLOAD, INSTALL, COPY OR OTHERWISE USE THE SOFTWARE (AS APPLICABLE). THIS LICENSE AGREEMENT REPRESENTS THE ENTIRE AGREEMENT CONCERNING THE SOFTWARE BETWEEN YOU AND RADWARE, AND SUPERSEDES ANY AND ALL PRIOR PROPOSALS, REPRESENTATIONS, OR UNDERSTANDINGS BETWEEN THE PARTIES. "YOU" MEANS THE NATURAL PERSON OR THE ENTITY THAT IS AGREEING TO BE BOUND BY THIS LICENSE AGREEMENT, THEIR EMPLOYEES AND THIRD PARTY CONTRACTORS. YOU SHALL BE LIABLE FOR ANY FAILURE BY SUCH EMPLOYEES AND THIRD PARTY CONTRACTORS TO COMPLY WITH THE TERMS OF THIS LICENSE AGREEMENT.

- 1. License Grant.** Subject to the terms of this Agreement, Radware hereby grants to you, and you accept, a limited, nonexclusive, nontransferable license to install and use the Software in machine-readable, object code form only and solely for your internal business purposes ("Commercial License"). If the Software is distributed to you with a software development kit (the "SDK"), then, solely with regard to the SDK, the Commercial License above also includes a limited, nonexclusive, nontransferable license to install and use the SDK solely on computers within your organization, and solely for your internal development of an integration or interoperation of the Software and/or other Radware Products with software or hardware products owned, licensed and/or controlled by you (the "SDK Purpose"). To the extent an SDK is

distributed to you together with code samples in source code format (the "Code Samples") that are meant to illustrate and teach you how to configure, monitor and/or control the Software and/or any other Radware Products, the Commercial License above further includes a limited, nonexclusive, nontransferable license to copy and modify the Code Samples and create derivative works based thereon solely for the SDK Purpose and solely on computers within your organization. The SDK shall be considered part of the term "Software" for all purposes of this License Agreement. You agree that you will not sell, assign, license, sublicense, transfer, pledge, lease, rent or share your rights under this License Agreement nor will you distribute copies of the Software or any parts thereof. Rights not specifically granted herein, are specifically prohibited.

2. **Evaluation Use.** Notwithstanding anything to the contrary in this License Agreement, if the Software is provided to you for evaluation purposes, as indicated in your purchase order or sales receipt, on the website from which you download the Software, as inferred from any time-limited evaluation license keys that you are provided with to activate the Software, or otherwise, then You may use the Software only for internal evaluation purposes ("Evaluation Use") for a maximum of 30 days or such other duration as may specified by Radware in writing at its sole discretion (the "Evaluation Period"). The evaluation copy of the Software contains a feature that will automatically disable it after expiration of the Evaluation Period. You agree not to disable, destroy, or remove this feature of the Software, and any attempt to do so will be a material breach of this License Agreement. During or at the end of the evaluation period, you may contact Radware sales team to purchase a Commercial License to continue using the Software pursuant to the terms of this License Agreement. If you elect not to purchase a Commercial License, you agree to stop using the Software and to delete the evaluation copy received hereunder from all computers under your possession or control at the end of the Evaluation Period. In any event, your continued use of the Software beyond the Evaluation Period (if possible) shall be deemed your acceptance of a Commercial License to the Software pursuant to the terms of this License Agreement, and you agree to pay Radware any amounts due for any applicable license fees at Radware's then-current list prices.
3. **Lab/Development License.** Notwithstanding anything to the contrary in this License Agreement, if the Software is provided to you for use in your lab or for development purposes, as indicated in your purchase order, sales receipt, the part number description for the Software, the Web page from which you download the Software, or otherwise, then You may use the Software only in your lab and only in connection with Radware Products that you purchased or will purchase (in case of a lab license) or for internal testing and development purposes (in case of a development license) but not for any production use purposes.
4. **Subscription Software.** If you licensed the Software on a subscription basis, your rights to use the Software are limited to the subscription period. You have the option to extend your subscription. If you extend your subscription, you may continue using the Software until the end of your extended subscription period. If you do not extend your subscription, after the expiration of your subscription, you are legally obligated to discontinue your use of the Software and completely remove the Software from your system.
5. **Feedback.** Any feedback concerning the Software including, without limitation, identifying potential errors and improvements, recommended changes or suggestions ("Feedback"), provided by you to Radware will be owned exclusively by Radware and considered Radware's confidential information. By providing Feedback to Radware, you hereby assign to Radware all of your right, title and interest in any such Feedback, including all intellectual property rights therein. With regard to any rights in such Feedback that cannot, under applicable law, be assigned to Radware, you hereby irrevocably waives such rights in favor of Radware and grants Radware under such rights in the Feedback, a worldwide, perpetual royalty-free, irrevocable, sub-licensable and non-exclusive license, to use, reproduce, disclose, sublicense, modify, make, have made, distribute, sell, offer for sale, display, perform, create derivative works of and otherwise exploit the Feedback without restriction. The provisions of this Section 5 will survive the termination or expiration of this Agreement.
6. **Limitations on Use.** You agree that you will not: (a) copy, modify, translate, adapt or create any derivative works based on the Software; or (b) sublicense or transfer the Software, or include the Software or any portion thereof in any product; or (b) reverse assemble, disassemble, decompile, reverse engineer or otherwise attempt to derive source code (or the

underlying ideas, algorithms, structure or organization) from the Software, in whole or in part, except and only to the extent: (i) applicable law expressly permits any such action despite this limitation, in which case you agree to provide Radware at least ninety (90) days advance written notice of your belief that such action is warranted and permitted and to provide Radware with an opportunity to evaluate if the law's requirements necessitate such action, or (ii) required to debug changes to any third party LGPL-libraries linked to by the Software; or (c) create, develop, license, install, use, or deploy any software or services to circumvent, enable, modify or provide access, permissions or rights which violate the technical restrictions of the Software; (d) in the event the Software is provided as an embedded or bundled component of another Radware Product, you shall not use the Software other than as part of the combined Product and for the purposes for which the combined Product is intended; (e) remove any copyright notices, identification or any other proprietary notices from the Software (including any notices of Third Party Software (as defined below)); or (f) copy the Software onto any public or distributed network or use the Software to operate in or as a time-sharing, outsourcing, service bureau, application service provider, or managed service provider environment. Notwithstanding the foregoing, if you provide hosting or cloud computing services to your customers, you are entitled to use and include the Software in your IT infrastructure on which you provide your services. It is hereby clarified that the prohibitions on modifying, or creating derivative works based on, any Software provided by Radware, apply whether the Software is provided in a machine or in a human readable form. Human readable Software to which this prohibition applies includes (without limitation) "Radware AppShape++ Script Files" that contain "Special License Terms". It is acknowledged that examples provided in a human readable form may be modified by a user.

7. **Intellectual Property Rights.** You acknowledge and agree that this License Agreement does not convey to you any interest in the Software except for the limited right to use the Software, and that all right, title, and interest in and to the Software, including any and all associated intellectual property rights, are and shall remain with Radware or its third party licensors. You further acknowledge and agree that the Software is a proprietary product of Radware and/or its licensors and is protected under applicable copyright law.
8. **No Warranty.** The Software, and any and all accompanying software, files, libraries, data and materials, are distributed and provided "AS IS" by Radware or by its third party licensors (as applicable) and with no warranty of any kind, whether express or implied, including, without limitation, any non-infringement warranty or warranty of merchantability or fitness for a particular purpose. Neither Radware nor any of its affiliates or licensors warrants, guarantees, or makes any representation regarding the title in the Software, the use of, or the results of the use of the Software. Neither Radware nor any of its affiliates or licensors warrants that the operation of the Software will be uninterrupted or error-free, or that the use of any passwords, license keys and/or encryption features will be effective in preventing the unintentional disclosure of information contained in any file. You acknowledge that good data processing procedure dictates that any program, including the Software, must be thoroughly tested with non-critical data before there is any reliance on it, and you hereby assume the entire risk of all use of the copies of the Software covered by this License. Radware does not make any representation or warranty, nor does Radware assume any responsibility or liability or provide any license or technical maintenance and support for any operating systems, databases, migration tools or any other software component provided by a third party supplier and with which the Software is meant to interoperate.

This disclaimer of warranty constitutes an essential and material part of this License.

In the event that, notwithstanding the disclaimer of warranty above, Radware is held liable under any warranty provision, Radware shall be released from all such obligations in the event that the Software shall have been subject to misuse, neglect, accident or improper installation, or if repairs or modifications were made by persons other than by Radware's authorized service personnel.

9. **Limitation of Liability.** Except to the extent expressly prohibited by applicable statutes, in no event shall Radware, or its principals, shareholders, officers, employees, affiliates, licensors, contractors, subsidiaries, or parent organizations (together, the "Radware Parties"), be liable for any direct, indirect, incidental, consequential, special, or punitive damages whatsoever relating to the use of, or the inability to use, the Software, or to your relationship with, Radware or any of the Radware Parties (including, without limitation, loss or disclosure of data or information,

and/or loss of profit, revenue, business opportunity or business advantage, and/or business interruption), whether based upon a claim or action of contract, warranty, negligence, strict liability, contribution, indemnity, or any other legal theory or cause of action, even if advised of the possibility of such damages. If any Radware Party is found to be liable to You or to any third-party under any applicable law despite the explicit disclaimers and limitations under these terms, then any liability of such Radware Party, will be limited exclusively to refund of any license or registration or subscription fees paid by you to Radware.

10. **Third Party Software.** The Software includes software portions developed and owned by third parties (the "Third Party Software"). Third Party Software shall be deemed part of the Software for all intents and purposes of this License Agreement; provided, however, that in the event that a Third Party Software is a software for which the source code is made available under an open source software license agreement, then, to the extent there is any discrepancy or inconsistency between the terms of this License Agreement and the terms of any such open source license agreement (including, for example, license rights in the open source license agreement that are broader than the license rights set forth in Section 1 above and/or no limitation in the open source license agreement on the actions set forth in Section 6 above), the terms of any such open source license agreement will govern and prevail. The terms of open source license agreements and copyright notices under which Third Party Software is being licensed to Radware or a link thereto, are included with the Software documentation or in the header or readme files of the Software. Third Party licensors and suppliers retain all right, title and interest in and to the Third Party Software and all copies thereof, including all copyright and other intellectual property associated therewith. In addition to the use limitations applicable to Third Party Software pursuant to Section 6 above, you agree and undertake not to use the Third Party Software as a general SQL server, as a stand-alone application or with applications other than the Software under this License Agreement.
11. **Term and Termination.** This License Agreement is effective upon the first to occur of your opening the package of the Product, purchasing, downloading, installing, copying or using the Software or any portion thereof, and shall continue until terminated. However, sections 5-15 shall survive any termination of this License Agreement. The Licenses granted under this License Agreement are not transferable and will terminate upon: (i) termination of this License Agreement, or (ii) transfer of the Software, or (iii) in the event the Software is provided as an embedded or bundled component of another Radware Product, when the Software is unbundled from such Product or otherwise used other than as part of such Product. If the Software is licensed on subscription basis, this Agreement will automatically terminate upon the termination of your subscription period if it is not extended.
12. **Export.** The Software or any part thereof may be subject to export or import controls under applicable export/import control laws and regulations including such laws and regulations of the United States and/or Israel. You agree to comply with such laws and regulations, and, agree not to knowingly export, re-export, import or re-import, or transfer products without first obtaining all required Government authorizations or licenses therefor. Furthermore, You hereby covenant and agree to ensure that your use of the Software is in compliance with all other foreign, federal, state, and local laws and regulations, including without limitation all laws and regulations relating to privacy rights, and data protection. You shall have in place a privacy policy and obtain all of the permissions, authorizations and consents required by applicable law for use of cookies and processing of users' data (including without limitation pursuant to Directives 95/46/EC, 2002/58/EC and 2009/136/EC of the EU if applicable) for the purpose of provision of any services.
13. **US Government.** To the extent you are the U.S. government or any agency or instrumentality thereof, you acknowledge and agree that the Software is a "commercial computer software" and "commercial computer software documentation" pursuant to applicable regulations and your use of the Software is subject to the terms of this License Agreement.
14. **Federal Acquisition Regulation (FAR)/Data Rights Notice.** Radware's commercial computer software is created solely at private expense and is subject to Radware's commercial license rights.
15. **Governing Law.** This License Agreement shall be construed and governed in accordance with the laws of the State of Israel.

16. **Miscellaneous.** If a judicial determination is made that any of the provisions contained in this License Agreement is unreasonable, illegal or otherwise unenforceable, such provision or provisions shall be rendered void or invalid only to the extent that such judicial determination finds such provisions to be unreasonable, illegal or otherwise unenforceable, and the remainder of this License Agreement shall remain operative and in full force and effect. In any event a party breaches or threatens to commit a breach of this License Agreement, the other party will, in addition to any other remedies available to, be entitled to injunction relief. This License Agreement constitutes the entire agreement between the parties hereto and supersedes all prior agreements between the parties hereto with respect to the subject matter hereof. The failure of any party hereto to require the performance of any provisions of this License Agreement shall in no manner affect the right to enforce the same. No waiver by any party hereto of any provisions or of any breach of any provisions of this License Agreement shall be deemed or construed either as a further or continuing waiver of any such provisions or breach waiver or as a waiver of any other provision or breach of any other provision of this License Agreement.

IF YOU DO NOT AGREE WITH THE TERMS OF THIS LICENSE YOU MUST REMOVE THE SOFTWARE FROM ANY DEVICE OWNED BY YOU AND IMMEDIATELY CEASE USING THE SOFTWARE.

COPYRIGHT © 2020, Radware Ltd. All Rights Reserved.