



N1 Service Provisioning System

4.1 Reference Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-4821
February 2004

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, the N1 logo, N1 and N1 Service Provisioning System, Java and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, le logo N1, N1 et N1 Service Provisioning System, Java et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



040212@7940



Contents

Preface 23

1 An Overview of N1 Service Provisioning System Software XML Schemas 27

Types of Plans	27
Types of Steps	28
Types of Components	28
Component References	29
Universal Install Path Format	30
Requirements for Locales and Character Sets	31
Pattern Matching	31
Variables and Parameter Passing	31
Call Compatibility	32
Install Compatibility	33
Boolean Operators	34
Attribute Types	38
Component Change Compatibility	40
Attribute and Child Element Ordering	46

2 The N1 Service Provisioning System Software Component Schema 47

Components	47
The component Element	47
component Element attributes	48
component Child Elements	51
The extends Element	51
extends Child elements	52

The type Element	52
type Element attributes	52
The component varList Element	52
component varList Child elements	53
component varList Inheritance and Overriding	53
The component var Element	53
component var Element attributes	53
var Inheritance and Overriding	54
The resourceRef Element	55
resourceRef Element attributes	55
resourceRef Child elements	56
resourceRef Inheritance and Overriding	56
The installSpec Element	57
installSpec Element attributes	57
The resource Element	59
resource Element attributes	59
The componentRefList Element	59
componentRefList Element attributes	60
componentRefList Child elements	60
componentRefList Inheritance and Overriding	60
The componentRef Element	61
The componentRef Element attributes	61
The componentRef Child elements	63
componentRef Inheritance and Overriding	63
The componentRef/argList Element	64
The componentRef/component Element	65
The installList Element	65
installList Child elements	65
installList Inheritance and Overriding	66
The installSteps Element	66
installSteps Element attributes	66
installSteps Child elements	67
installSteps Inheritance and Overriding	67
The paramList Element	68
paramList Child elements	69
The param Element	69
param Element attributes	69
The local varList Element	70

local varList Child elements	70
The local var Element	71
local var Element attributes	71
uninstallList Element	71
uninstallList Child elements	72
uninstallList Inheritance and Overriding	72
The uninstallSteps Element	72
uninstallSteps Element attributes	72
uninstallSteps Child elements	73
uninstallSteps Inheritance and Overriding	74
The dependantCleanup Block Element	74
The snapshotList Element	74
snapshotList Child elements	75
snapshotList Inheritance and Overriding	75
The snapshot Element	75
snapshot Element attributes	76
snapshot Child elements	77
snapshot Inheritance and Overriding	77
The snapshot prepare Child element	78
prepare Child elements	78
The snapshot capture Child element	78
capture Child elements	78
The capture addFile Child element	79
addFile Element attributes	79
The capture addSnapshot Child element	80
addSnapshot Element attributes	81
addSnapshot Child elements	81
The capture addResource Child element	81
The snapshot cleanup Child element	82
The snapshot cleanup Child elements	82
The controlList Element	82
controlList Child elements	82
controlList Inheritance and Overriding	83
The control Element	83
control Element attributes	83
control Child elements	84
control Inheritance and Overriding	84
The diff Element	85

diff Child elements	85
diff Inheritance and Overriding	85
The ignore Element	85
ignore Element attributes	85
Installed Component Targeters	86
The systemService Installed Component Targeter	87
installedComponent Targeter attributes	87
systemService Targeter attributes	88
The systemType Installed Component Targeter	88
The installedComponent Installed Component Targeter	89
systemType Targeter attributes	89
The thisComponent Installed Component Targeter	89
The superComponent Installed Component Targeter	90
The nestedRef Installed Component Targeter	90
nestedRef Targeter Attributes	91
The allNestedRefs Installed Component Targeter	91
The toplevelRef Installed Component Targeter	92
toplevelRef Targeter Attributes	92
The dependee Installed Component Targeter	93
dependee Targeter Attributes	93
The allDependants Installed Component Targeter	93
allDependants Targeter Attributes	94
Repository Component Targeters	94
The component Repository Component Targeter	94
component Targeter Attributes	95
The thisComponent Repository Component Targeter	95
The superComponent Repository Component Targeter	95
The nestedRef Repository Component Targeter	96
nestedRef Targeter Attributes	96
The allNestedRefs Repository Component Targeter	96
The toplevelRef Repository Component Targeter	97
toplevelRef Targeter Attributes	97
Component Install-only Steps	98
createDependency Step	98
createDependency Step attributes	98
createDependency Step Child elements	99
createDependency Step Uninstall Implications	99
createDependency Step Reinstall Implications	99

createDependency Step Naming Conventions	100
createSnapshot Step	100
createSnapshot Step attributes	101
install Step	101
Simple Component Install-only Steps	101
deployResource Step	101
Component Uninstall-only Steps	102
uninstall Step	102
Simple Component Uninstall-only Steps	103
undeployResource Step	103
3 The N1 Service Provisioning System Software Plan Schema	105
The executionPlan Element	105
executionPlan Element attributes	105
executionPlan child elements	106
The Plan paramList Element	106
Plan paramList Element child elements	107
The Plan param Child Element	107
Plan param Child Element attributes	107
The Plan varList Element	108
Plan varList Element	108
The Plan var Child Element	109
Plan var Child Element attributes	109
The simpleSteps Element	109
simpleSteps Element attributes	109
Child Elements of the simpleSteps Elements	110
The compositeSteps Element	110
Child Elements of the compositeSteps Element	110
Child Elements of the compositeSteps Element	111
Shared Steps	111
call Step	111
call Step attributes	111
call Step child elements	112
The argList Child Element	112
argList Child Element attributes	113
CheckDependency Step	113
checkDependency Step child elements	113

execJava Step	113
execJava Step attributes	114
execJava Child elements	114
execNative Step	115
execNative Step attributes	115
execNative Element child elements	116
The env Child Element	117
background element	117
outputFile element	118
errorFile element	118
The inputText Child Element	119
inputFile Child Element	119
The exec Child Element	120
The shell Child Element	121
The successCriteria Element	122
if Step	124
if Step Child Elements	124
if Step Example	124
condition Element	124
then Element	125
else Element	125
pause Step	125
pause Step attributes	125
processTest Step	125
processTest Step attributes	126
raise Step	126
raise Step attributes	126
raise Step Example	126
reboot Step	127
reboot Step attributes	127
retarget Step	127
retarget Step attributes	128
retarget Step Child Elements	128
retarget Step host Attribute	128
retarget Step Execution Semantics	128
retarget Step Example	129
The sendCustomEvent Step	130
sendCustomEvent Step attributes	130

transform Step	130
transform Step attributes	131
transform Element child elements	131
The stylesheet transform Child Element	131
The subst transform Child Element	132
subst Element attributes	132
An Example of the subst Element	133
The source transform Child Element	133
try Step	134
try Step Child Elements	135
try Step Examples	136
block Element	137
catch Element	137
finally Element	137
urlTest Step	137
Attributes of the urlTest Step	138
Composite Plan-only Steps	138
execSubplan Step	138
execSubplan Step attributes	138
execSubplan Step child elements	139
inlineSubplan Step	139
inlineSubplan Step attributes	140
inlineSubplan Element child elements	140
Simple Plan-only Steps	141
install Step	141
install Step attributes	141
install Step child elements	142
uninstall Step	142
uninstall Step attributes	142
uninstall Step child elements	142
4 The N1 Service Provisioning System Software Command-Line Interface	143
The Structure of CLI Commands	144
Invoking CLI Commands	144
Working with the CLI	145
Authentication by Username and Password or Session ID	145
Listing the Commands	145

	Getting Help with the CLI	145
	CLI Syntax	146
	Selecting the Output Format of a CLI Command	146
	Passing Output Data to a File	148
	Passing Data to an Individual CLI Argument	149
	Converting the Output Format	149
	Passing Arguments to the CLI from a File	150
	Descriptions and Multi-word Names	150
	ID Substitution Using NM:	150
	Showing and Hiding Objects	151
	Working with CLI Commands	152
	Required Arguments and Optional Arguments	152
	The cr_cli Tool	153
	The cr_clij Tool	154
	Jython Scripts	154
	CLI Input Types	154
5	cat: Commands for Managing Categories	161
	Overview of the cat Commands	161
	cat.add	162
	cat.del	162
	cat.mod	162
	cat.la	163
6	cdb: CLI Commands for Managing Components	165
	Overview of the cdb Commands	165
	cdb.c: Managing Components	166
	cdb.c.ci	166
	cdb.c.co	167
	cdb.c.la	167
	cdb.c.lo	168
	cdb.c.lv	168
	cdb.c.mod	168
	cdb.c.sc	169
	cdb.c.sc	169
	cdb.c.del	170
	cdb.ic: Managing Installed Components	170

	cdb.ic.lbc	171
	cdb.ic.lbh	171
	cdb.ic.vs.lo	171
	cdb.vs: Managing Variable Settings	172
	cdb.vs.add	172
	cdb.vs.del	173
	cdb.vs.imp	173
	cdb.vs.la	173
	cdb.vs.lo	174
	cdb.vs.mod	174
	cdb.ssr: System Service Ref Commands	175
	cdb.ssr.add	175
	cdb.ssr.mod	176
	cdb.ssr.del	176
	cdb.ssr.lo	176
	cdb.ssr.la	177
	cdb.ctr: Component Type Commands	177
	cdb.ctr.add	177
	cdb.ctr.mod	178
	cdb.ctr.del	179
	cdb.ctr.lo	179
	cdb.ctr.la	179
7	cfg: CLI Commands for Performing Config-Generation	181
	Overview of the <code>cfg</code> Command	181
	<code>cfg.gen</code>	181
8	cmp: CLI Commands for Running Comparisons	183
	Overview of the <code>cmp</code> Commands	183
	<code>cmp.dj.add</code>	184
	<code>cmp.dj.del</code>	184
	<code>cmp.dj.la</code>	184
	<code>cmp.dj.lo</code>	185
	<code>cmp.ds.add</code>	185
	<code>cmp.ds.la</code>	187
	<code>cmp.ds.lo</code>	187
	<code>cmp.ds.del</code>	187

cmp.ds.mod 188
cmp.ds.sc 189

9 hdb: CLI Commands for Managing Hosts 191

Introduction	191
hdb.a: Managing Application Instances	192
hdb.a.add	192
hdb.a.del	193
hdb.a.la	193
hdb.a.lo	194
hdb.a.mod	194
hdb.a.clear	195
hdb.h: Managing Hosts	195
hdb.h.add	195
hdb.h.del	196
hdb.h.la	196
hdb.h.lo	196
hdb.h.lq	197
hdb.h.mod	197
hdb.hr: Managing Host Searches	198
Overview	198
hdb.hr.add	199
hdb.hr.del	199
hdb.hr.la	199
hdb.hr.le	200
hdb.hr.lo	200
hdb.hr.mod	200
hdb.hs: Managing Host Sets	201
hdb.hs.add	201
hdb.hs.del	202
hdb.hs.la	202
hdb.hs.le	203
hdb.hs.lo	203
hdb.hs.mod	203
hdb.ht: Managing Host Types	204
Overview	204
hdb.ht.add	204

	hdb.ht.del	205
	hdb.ht.la	205
	hdb.ht.lo	205
	hdb.ht.mod	206
10	net: CLI Commands for Performing Network Operations	207
	Overview of the net Commands	207
	net.gencfg	208
	net.traceroute	208
11	pdb: CLI Commands for Managing Plans	209
	Overview of the pdb Commands	209
	pdb.p.ci	209
	pdb.p.co	210
	pdb.p.genplan	210
	pdb.p.la	211
	pdb.p.lo	211
	pdb.p.del	211
	pdb.p.lv	211
	pdb.p.sh	212
	pdb.p.sc	212
12	pe: CLI Commands for Running Plans	213
	Overview of the pe Commands	213
	pe.h.prep	214
	pe.p.en	214
	pe.p.la	214
	pe.p.lo	215
	pe.p.del	215
	pe.p.lp	216
	pe.p.run	216
	pe.p.stop	218
	pe.pi.lo	218
13	CLI Commands for Managing Resources	219
	Overview of the cdb Commands	219

- cdb.rsrc.ci 220
- cdb.rsrc.cib 222
- cdb.cj.la 227
- cdb.cj.lo 227
- cdb.cj.stop 228
- cdb.rsrc.co 228
- cdb.rsrc.rci 229

14 rule: CLI Commands for Notifications 231

- Overview of the rule Commands 231
- rule.add 231
- rule.del 232
- rule.la 233
- rule.lo 233
- rule.mod 233

15 udb: CLI Commands for Managing Users and Groups 235

- Overview of the udb Commands 235
- udb.g: Managing User Groups 236
 - udb.g.add 236
 - udb.g.del 237
 - udb.g.la 237
 - udb.g.lo 237
 - udb.g.lp 237
 - udb.g.lu 238
 - udb.g.mod 238
- udb.u: Managing User Accounts 239
 - udb.u.add 239
 - udb.u.cp 239
 - udb.u.la 240
 - udb.u.lo 240
 - udb.u.lp 240
 - udb.u.mod 241
- udb.sv: Managing Session Variables 241
 - udb.sv.add 242
 - udb.sv.del 242
 - udb.sv.la 243

	udb.sv.lo	243
	udb.sv.mod	243
	Authentication Commands	244
	udb.login	244
	udb.logout	244
	udb.whoami	244
	udb.p: Commands for Managing Permissions	245
	udb.p.la	245
	udb.p.lo	245
16	Configuration Generation	247
	Adding Substitution Variable Definitions to Components	247
	Variable Substitution Values	248
	Generation Context	248
	Types of Variables Available for Substitution	248
	Simple Substitution Variables	249
	External Substitution Variables	249
	External Component Substitution Reference Resolution	253
	Session Variables Substitution Variables	254
	Target Substitution Variables	255
	Predefined Host Substitution Shorthands	257
	Using Substitution Variables	257
	Escape Sequences Within Files With Substitution Variable Definitions	262
	Predefined Substitution Variables for Components and Configuration Files	263

Tables

TABLE 1-1	Examples of Component Attributes	29
TABLE 1-2	Component Versions Referenced by Attribute Combination	30
TABLE 4-1	Syntax of a CLI Command.	146
TABLE 4-2	CLI Format Options	147
TABLE 4-3	Command Output Formats for CLI Command Families	147
TABLE 4-4	Subsystems and CLI Prefixes	152
TABLE 4-5	O/R Notation for Arguments	152
TABLE 4-6	Syntax of CLI Input Types	155
TABLE 5-1	Summary of cat Commands	161
TABLE 5-2	Arguments and Result for the cat.add Command	162
TABLE 5-3	Argument for the cat.del Command	162
TABLE 5-4	Arguments and Result for the cat.mod Command	163
TABLE 5-5	Result for the cat.la Command	163
TABLE 6-1	Sets of Commands for Managing Components	165
TABLE 6-2	CLI Commands for Managing Components	166
TABLE 6-3	Arguments and Result for the cdb.c.ci Command	167
TABLE 6-4	Argument and Result for the cdb.c.co Command	167
TABLE 6-5	Arguments and Result for the cdb.c.la Command	167
TABLE 6-6	Arguments and Result for the cdb.c.lo Command	168
TABLE 6-7	Argument and Result for the cdb.c.lv Command	168
TABLE 6-8	Arguments and Result for the cdb.c.mod Command	168
TABLE 6-9	Arguments and Result for the cdb.c.sc Command	169
TABLE 6-10	Arguments and Result for the cdb.c.sc Command	170
TABLE 6-11	Arguments for the cdb.c.del Command	170
TABLE 6-12	CLI Commands for Managing Installed Components	170
TABLE 6-13	Argument and result for the cdb.ic Command	171

TABLE 6-14	Argument and Result for the cdb.ic.lbh Command	171
TABLE 6-15	Argument and Result for the cdb.ic.vs.lo Command	172
TABLE 6-16	CLI Commands for Managing Variable Settings	172
TABLE 6-17	Arguments and Result for the cdb.vs.add Command	172
TABLE 6-18	Argument for the cdb.vs.del Command	173
TABLE 6-19	Arguments for the cdb.vs.imp Command	173
TABLE 6-20	Argument and Result for the cdb.vs.la Command	174
TABLE 6-21	Argument and Result for the cdb.vs.lo Command	174
TABLE 6-22	Arguments and Result for the cdb.vs.mod Command	174
TABLE 6-23	CLI Commands for System Service Ref	175
TABLE 6-24	Argument and Result for the cdb.ssr.add Command	175
TABLE 6-25	Argument and Result for the cdb.ssr.mod Command	176
TABLE 6-26	Argument and Result for the cdb.ssr.del Command	176
TABLE 6-27	Argument and Result for the cdb.ssr.lo Command	177
TABLE 6-28	Argument and Result for the cdb.ssr.lo Command	177
TABLE 6-29	CLI Commands for System Service Ref	177
TABLE 6-30	Argument and Result for the cdb.ctr.add Command	178
TABLE 6-31	Argument and Result for the cdb.ctr.mod Command	178
TABLE 6-32	Argument and Result for the cdb.ctr.del Command	179
TABLE 6-33	Argument and Result for the cdb.ctr.lo Command	179
TABLE 6-34	Argument and Result for the cdb.ctr.la Command	180
TABLE 7-1	Summary of the cfg Commands	181
TABLE 7-2	Arguments for the cfg.gen Command	181
TABLE 8-1	Summary of cmp Commands	183
TABLE 8-2	Argument and Result for the cmp.dj.add Command	184
TABLE 8-3	Argument and Result for the cmp.dj.del Command	184
TABLE 8-4	Argument and Result for the cmp.dj.la Command	184
TABLE 8-5	Arguments for the cmp.dj.lo	185
TABLE 8-6	Arguments and Result for cmp.ds.add	185
TABLE 8-7	Argument and Result for the cmp.ds.la Command	187
TABLE 8-8	Argument and Result for the cmp.ds.lo	187
TABLE 8-9	Argument and Result for the cmp.ds.del	187
TABLE 8-10	Argument and Result for the cmp.ds.mod	188
TABLE 8-11	Argument and Result for the cmp.ds.sc Command	189
TABLE 9-1	Sets of Commands for Managing Hosts	191
TABLE 9-2	Summary of hdb.a Commands	192
TABLE 9-3	Arguments and Result for the hdb.a.add Command	192
TABLE 9-4	Argument for the hdb.a.del Command	193

TABLE 9-5	Result Returned by the hdb.a.la command	193
TABLE 9-6	Argument and Results of the hdb.a.lo Command	194
TABLE 9-7	Arguments and Result for the hdb.a.mod Command	194
TABLE 9-8	Arguments and Result for the hdb.a.clear Command	195
TABLE 9-9	Summary of hdb.h Commands	195
TABLE 9-10	Arguments and Results for the hdb.h.add Command	195
TABLE 9-11	Arguments and Results for the hdb.h.del Command	196
TABLE 9-12	Argument and Result for the hdb.h.la Command	196
TABLE 9-13	Argument and Result for the hdb.h.lo Command	197
TABLE 9-14	Arguments and Result for the hdb.h.lq Command	197
TABLE 9-15	Arguments and Result of the hdb.h.mod Command	197
TABLE 9-16	Summary of hdb.hr Commands	198
TABLE 9-17	Arguments and Result for the hdb.hr.add Command	199
TABLE 9-18	Arguments and Result for the hdb.hr.del Command	199
TABLE 9-19	Argument and Result for the hdb.hr.la Command	200
TABLE 9-20	Arguments and Result for the hdb.hr.le Command	200
TABLE 9-21	Argument and Result for the hr.hr.lo	200
TABLE 9-22	Arguments and Results for the hdb.hr.mod Command	201
TABLE 9-23	Summary of hdb.hs Commands	201
TABLE 9-24	Arguments and Result for the hdb.hs.add Command	202
TABLE 9-25	Arguments and Result for the hdb.hs.del Command	202
TABLE 9-26	Argument and Result for the hdb.hs.la Command	202
TABLE 9-27	Arguments and Results for the hdb.hs.le Command	203
TABLE 9-28	Argument and Result for the hdb.hs.lo Command	203
TABLE 9-29	Arguments and Result for the hdb.hs.mod Command	203
TABLE 9-30	Summary of hdb.ht Commands	204
TABLE 9-31	Arguments and Result for the hdb.ht.add Command	205
TABLE 9-32	Arguments and Result for the hdb.ht.del Command	205
TABLE 9-33	Argument and Result for the hdb.ht.la Command	205
TABLE 9-34	Argument and Result for the hdb.ht.lo Command	206
TABLE 9-35	Arguments and Result for the hdb.ht.mod Command	206
TABLE 10-1	Summary of the net Commands	207
TABLE 10-2	Argument for the net.gencfg Command	208
TABLE 10-3	Argument and Result for the net.traceroute Command	208
TABLE 11-1	Summary of the pdb Commands	209
TABLE 11-2	Arguments and Result for the pdb.p.ci	210
TABLE 11-3	Argument for the pdp.p.co Command	210
TABLE 11-4	Arguments for the pdb.p.genplan Command	210

TABLE 11-5	Arguments and Result for the pdb.p.la Command	211
TABLE 11-6	Argument and Result for the pdb.p.lo Command	211
TABLE 11-7	Argument and Result for the pdb.p.del Command	211
TABLE 11-8	Argument and Result of the pdb.p.lv Command	212
TABLE 11-9	Arguments for the pdb.p.sh Command	212
TABLE 11-10	Arguments for the pdb.p.sc Command	212
TABLE 12-1	Summary of the pe Commands	213
TABLE 12-2	Argument and Result of the pe.h.prep Command	214
TABLE 12-3	Arguments and Result for the pe.p.en Command	214
TABLE 12-4	Arguments and Result for the pe.p.la Command	215
TABLE 12-5	Arguments for the pe.p.lo Command	215
TABLE 12-6	Arguments for the pe.p.del Command	216
TABLE 12-7	Argument and Result for the pe.p.lp Command	216
TABLE 12-8	Arguments and Result for the pe.p.run Command	216
TABLE 12-9	Argument and Result for the pe.p.stop Command	218
TABLE 12-10	Argument and Result of the pe.pi.lo Command	218
TABLE 13-1	Summary of the cdb.rsrc Commands	219
TABLE 13-2	Arguments and Result for the cdb.rsrc.ci Command	220
TABLE 13-3	Arguments and Result of the cdb.rsrc.cib	222
TABLE 13-4	Syntax of a Line in a Batch File	223
TABLE 13-5	Names for Platforms	225
TABLE 13-6	Result of the cdb.cj.la Command	227
TABLE 13-7	Argument and Result of the cdb.cj.lo Command	227
TABLE 13-8	Argument and Result of the rsrc.cj.stop	228
TABLE 13-9	Arguments and Result for the cdb.rsrc.co	228
TABLE 13-10	Argument and Result for the cdb.rsrc.rci Command	229
TABLE 14-1	Summary of the rule Commands	231
TABLE 14-2	Arguments and Result for the rule.add Command	232
TABLE 14-3	CriteriaShorthand for the -tf Argument	232
TABLE 14-4	Argument for the rule.del Command	233
TABLE 14-5	Result of the rule.la Command	233
TABLE 14-6	Argument and Result for the rule.lo Command	233
TABLE 14-7	Arguments and Result for the rule.mod Command	233
TABLE 15-1	Sets of Commands for User Accounts, Groups, and Logins	235
TABLE 15-2	Summary of udb.g Commands	236
TABLE 15-3	Arguments and Result for the udb.g.add Command	236
TABLE 15-4	Argument for the udb.g.del Command	237
TABLE 15-5	Result for the udb.g.la Command	237

TABLE 15-6	Argument and Result for the udb.g.lo Command	237
TABLE 15-7	Argument and Result for the udb.g.lp	238
TABLE 15-8	Argument and Result for the udb.g.lu Command	238
TABLE 15-9	Arguments and Result for the udb.g.mod Command	238
TABLE 15-10	Summary of udb.g Commands	239
TABLE 15-11	Arguments and Result for the udb.u.add Command	239
TABLE 15-12	Arguments for the udb.u.cp Command	240
TABLE 15-13	Argument and Result for the udb.u.la Command	240
TABLE 15-14	Argument and Result for the udb.u.lo Command	240
TABLE 15-15	Argument/Result for the udb.u.lp Command	241
TABLE 15-16	Argument/Result for the udb.u.mod Command	241
TABLE 15-17	Summary of udb.sv Commands	241
TABLE 15-18	Arguments and Result for the udb.sv.add Command	242
TABLE 15-19	Arguments for the udb.sv.del Command	243
TABLE 15-20	Argument and Result for the udb.sv.la Command	243
TABLE 15-21	Argument and Result for the udb.sv.lo Command	243
TABLE 15-22	Argument/Result for the udb.sv.mod Command	243
TABLE 15-23	Result of the udb.login Command	244
TABLE 15-24	Result of the udb.whoami Command	245
TABLE 15-25	Summary of the udb.p Commands	245
TABLE 15-26	Result for the udb.p.la Command	245
TABLE 15-27	Argument and Result for the udb.p.lo Command	245

Preface

The *N1 Service Provisioning System 4.1 Reference Guide* describes how to model and manage applications, how to use the command-line interface, and how to use many of the product's commands.

Who Should Use This Book

This book is for advanced system administrators who are performing the initial configuration of the N1 Service Provisioning System software and for system administrators who are responsible for maintaining the system.

This document is intended for experienced system administrators with extensive knowledge of Sun™ software and hardware. Do not use this document as a planning or pre-sales guide. You should have already determined your system requirements and purchased the appropriate equipment and software before reading this document.

How This Book Is Organized

In the *N1 Service Provisioning System 4.1 Reference Guide*, the first three chapters describe the XML schemas that you will use to create and manage components and plans. The remaining thirteen chapters of the reference guide describe the command-line interface and the commands that you will use to manage the N1 Service Provisioning System software

Related Books

The following books provide conceptual information or procedures to administer the N1 Service Provisioning System software. If you plan to use this documentation in a hardcopy format, ensure that you have the following books available for your reference.

- *N1 Service Provisioning System 4.1 Installation Guide*
- *N1 Service Provisioning System 4.1 User's Guide*
- *N1 Service Provisioning System 4.1 Release Notes*

Accessing Sun Documentation Online

The docs.sun.comSM Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is <http://docs.sun.com>.

Ordering Sun Documentation

Sun Microsystems offers select product documentation in print. For a list of documents and how to order them, see "Buy printed documentation" at <http://docs.sun.com>.

Typographic Conventions

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name%</code> su Password:
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type rm <i>filename</i> .
<i>AaBbCc123</i>	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You must be <i>root</i> to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	<code>machine_name%</code>
C shell superuser prompt	<code>machine_name#</code>
Bourne shell and Korn shell prompt	<code>\$</code>
Bourne shell and Korn shell superuser prompt	<code>#</code>

An Overview of N1 Service Provisioning System Software XML Schemas

The plan and component language or XML schema defines the syntax and semantics for the control structures used to model and manage applications within the N1™ Service Provisioning System software.

A component is a logical grouping of resources (files, directories, database records, etc.) files, that defines an application. The component also includes sequences of instructions used to compare, install, uninstall and control the application. An example of a control is a series of instructions used to “startup” or “shutdown” the application once installed. Possible instructions also include tests for pre and post conditions such as dependency checks on other components and verification that a particular process is running. Finally, the component may declare substitution variables for use within the component itself and any resources it contains as placeholders for values that will be substituted when the plan is run.

A plan is a sequence of instructions typically used to manipulate one or more components. For example, a plan may instruct the provisioning software to install three components and initiate the “startup” control on another. A plan can also be defined as a sequence of other plans, which allows common instruction sequences to be shared between multiple plans.

Types of Plans

The N1 Service Provisioning System software distinguishes between two kinds of plans.

- Composite plan (containing only subplans)
- Simple plan (collection of simple steps that can’t call another subplan)

XML Schema enforces that restriction, so that you can either have a top-level plan with only calls to other subplans, or you can have a simple plan with various simple steps but no calls to other subplans. The distinction is important because the steps contained in a simple plan all execute on the same set of target hosts, whereas the steps of a composite plan may execute on different sets of target hosts, using one set of target hosts for each simple plan recursively contained in the composite plan.

Types of Steps

There are also three kinds of steps that are available with the N1 Service Provisioning System software.

- Steps that can be called only from within a component. This category is further divided into steps that can only be called from install blocks, and steps that can only be called from uninstall blocks..
 - Steps that can be called only from a plan. This category is further divided into steps that can be called only from composite plans, and steps that can be called only from simple plans.
 - Steps that can be called from either a plan or a component.
-

Types of Components

Like plans, components come in two varieties, composite and simple. A simple component is a component that contains a single resource, but no references to other components. A composite component is a component that references other components, but does not contain any resources. A composite component may reference both composite and simple components.

A composite component contains the components that it references. The referenced components are referred to as contained components, and the referencing component is referred to as the containing or container component.

A composite component declares whether each of its contained components should be installed as top-level or nested components. If a contained component is installed as top-level, then it can be used by any other component just as if it had been directly installed by a plan. However, if a contained component is installed as nested, then its services are only available to the containing component. Logically, a nested contained component defines a finer-grained unit of functionality required by the containing component, but not otherwise useful to other components. Whereas a top level contained component defines services that will be used by the containing component, but may also be used by other components.

Note – Composite components contain only references to other components, but not the components themselves. The referenced components are other existing components that are updated and managed independent of the containing component. A component may be referenced by any number of composite components. A component's name is not affected by whether or not it is referenced by other composite component; name conflicts are resolved using paths.

When used within this document, the terms derived component, child component, and parent component refer to component inheritance relationships rather than component composition relationships.

Component References

There are a number of steps that reference components. These steps can be divided into two categories:

- Steps that reference components that have yet to be installed:
 - Install
- Steps that reference components that are already installed:
 - Uninstall
 - Call
 - CheckDependency
 - CreateDependency
 - AddSnapshot

Steps that reference components that have yet to be installed need only specify the component name and an optional version. However, steps that reference an installed component may additionally specify the install path in which the desired component is located. The installPath is needed in the case of installed component because the same component may be installed multiple times on the same machine.

For example, assume the "apache" component is installed on a host with the following attributes:

TABLE 1-1 Examples of Component Attributes

Component Instance	Install Path	Version	Install Date
A	/opt	1.3	6/1/01 5:00 PM

TABLE 1-1 Examples of Component Attributes (Continued)

B	/usr/local	1.4	6/1/01 6:00 PM
C	/opt	1.2	6/2/01 5:00 PM
D	/usr/local/bin	1.4	6/3/01 5:00 PM
E	/export	1.1	6/4/01 5:00 PM

Table 1-2 shows which installed component is referenced when various combinations of `installPath` and build version are supplied.

TABLE 1-2 Component Versions Referenced by Attribute Combination

Install Path	Version	Version Op	Result	Explanation
None	None	None	E	The most recently installed component on the target host is used, regardless of build and installPath.
/opt	None	None	C	The most recently installed component in the named install path is used regardless of build.
/usr/bin	None	None	ERROR	No component is installed in the named path.
None	1.4	=	D	The most recently installed component with the specified build is chosen, regardless of install path.
None	1.5	Any	ERROR	No component is installed with the named build.
/usr/local	1.4	=, >=	B	The component with the install path and build is chosen.
/usr/local	1.2	=	ERROR	No such build at that install path.
/usr/local	1.2	>, >=	B	Path, version and version operator match
/opt	3	Any	ERROR	If two or more components with the same name are installed in the same path, the most recently installed component effectively overwrites any other components installed earlier. Components installed at an earlier date cannot be accessed, even if named directly.

Universal Install Path Format

An install path may be specified within an installed component reference, as described in the previous section. In these cases, the install path attribute specified is converted to universal format prior to resolving the installed component reference. In universal format, all occurrences of the master server's OS specific path separator within the install path are replaced by the `"/"`, and any trailing `"/"` is dropped. An exception is made for the root install path (`"/"`), which is not converted to the empty path.

When the Master Server is running on UNIX based systems, the affect of this conversion is minimal. Install paths that end with `"/"` will be treated as if no `"/"` is included. For example, both `"/opt/apache/"` and `"/opt/apache"` can be used to refer to the component installed in directory `"/opt/apache"`.

This conversion takes place because the install path of the installed component is also stored in universal format.

Requirements for Locales and Character Sets

The N1 Service Provisioning System software has been internationalized to install and run in localized environments. You will need to adhere to the following requirements if you want to run the software in a localized environment.

- All applications must be run in the same locale or in locales that are equivalent. The Remote Agent, Local Distributors, and CLI Client must run in the same locale as the Master Server.
- The software accepts only ASCII characters for file names, directory names, and other input.

Pattern Matching

A number of attributes of elements within a component or plan may contain regular expression patterns. Unless otherwise specified, these patterns are glob-style patterns rather than fully generic regular expressions. This means that `"*"` is used to match zero or more characters and `"?"` is used to match exactly one character. Characters listed within brackets (`[]`) specify a range of possible characters that may match exactly one character.

Variables and Parameter Passing

Both plans and components may declare variables for use by their steps. Component variables are evaluated and bound at the time the component is installed. Therefore, if a step within a component control block references a component-scoped variable, the

value used will be the value of that variable at the time the component was installed. Plan variables are evaluated and bound each time the plan is run. Therefore, if a step within a plan references a plan variable, the value used will be the value defined at the time of the plan run, and may vary from one run to another.

A plan may declare both parameters and variables. The value of a variable is defined at the point of declaration based on the values of other variables and constants. A parameter is a special kind of variable whose value is defined by the caller. In the case of a top-level plan, the caller is the user who initiates the plan run. For each parameter declared by the plan, the user enters the desired value for that parameter prior to running the plan. When a plan is invoked as the result of an <execSubplan> call, the plan containing the <execSubplan> call is responsible for explicitly passing values for each of the parameters declared by the called plan.

It is also possible for install, uninstall, snapshot, and control blocks to declare parameters and local variables. As with plans, local variable values are locally defined, and parameter values are defined based on the values passed by the caller of the block. Both may vary each time a plan is run.

There are no facilities for re-assigning the value of a variable or parameter.

Call Compatibility

Component B is said to be call compatible with component A if uses of A can be safely replaced with uses of B in the following cases:

- A call to a control block of A.
- A call to an uninstall block of A.
- A call to a snapshot block of A (using addSnapshot).
- A check for dependency on A (using checkDependency or createDependency).
- A reference to a variable of A (using config gen :[component:AB:var]).

Call compatibility basically means API/interface compatibility.

Note – It is often the case that B and A are different versions of components in the same version tree. However, it is also possible for B to be in a distinct version tree if B is an instance of A.

The provisioning software enforces call compatibility for components that serve as system services. Specifically, when a system service is updated to reference a new component, we enforce that the new component is call compatible with the original component. This allows us to ensure that clients of the system service will continue to function properly when the system service is upgraded.

The provisioning software will also optionally verify call compatibility when resolving components referenced by certain installed component targeters. See the "Installed Component Targeters" section for details.

There is no requirement that a component be call compatible with earlier versions of itself, though it is generally recommended whenever possible.

Install Compatibility

Component B is said to be install compatible with component A if B is call compatible with A, and uses of A can be safely replaced with uses of B in the following cases:

- A call to an install block of A.
- A component extending from A.
- A component containing a reference to A.

Install compatibility basically means structural compatibility.

The constraints imposed by install compatibility mean that any existing installed component can be safely replaced by another install compatible component without having to modify our data structures regarding how the original was installed. Call compatibility is a much weaker statement, as the call compatible component may need to be reinstalled over the original component to properly update our data structures.

Note – For install compatibility to hold, B and A must belong to the same version tree. They may not be components from two distinct version trees.

The provisioning software only enforces install compatibility for components that serve as types. Specifically, when a component type is updated to reference a new version of a component, the new version is install compatible with the original version. Therefore, install compatible updates can be made to component types without having to rebuild and reinstall all existing components derived from that type. As a consequence, if a change is made to a component type that is not install compatible, the client will need to encode that change as a new, distinct component version tree with its own type name. In such cases, the new component type can maintain call compatibility with the original by extending from the original component type. To easily identify the relationship between types, use a versioning system to encode component type names. For example, the component names, EJB-1.0 and EJB-1.1 easily identifies EJB-1.1 as a later version of the EJB-1.0 component type.

Install compatible implies call compatible, but the reverse does not hold. It also follows that not call compatible implies not install compatible.

Boolean Operators

This section lists elements that serve as boolean operators, and appear within the condition of an `<if>` step. Boolean operators have the property that they evaluate to true or false.

istruer Boolean Operator

A boolean operator used to determine if a particular value is true. It contains a single attribute "value" and has no child elements. It evaluates to true if and only if the value of the "value" attribute case-insensitive equals "true".

istruer Boolean Operator Attributes

Name	Type	Required	Configurable	Description
value	string	Yes	Yes	The value to compare against "true".

istruer Boolean Operator Examples

Element	Result
<code><istruer value="True"/></code>	true
<code><istruer value="yes"/></code>	false
<code><istruer value=":[var]"/></code>	true if the value of var is true

equals Boolean Operator

A boolean operator used to determine if a particular value is equal to another value. It contains attributes "value1", "value2" and "exact", and has no child elements. It evaluates to true if and only if the values of the "value1" and "value2" attribute are equal. If "exact" is true, then they must be case-sensitive equals, otherwise they must be case-insensitive equals.

Note – `<istruer value="..."/>` is a syntactic shorthand for `<equals value1="..." value2="true"/>`.

equals Boolean Operator Attributes

Name	Type	Required	Configurable	Description
value1	string	Yes	Yes	A value to be compared.
value2	string	Yes	Yes	The other value to be compared.
exact	boolean	No	No	True if a case-sensitive match should be performed, false otherwise. Defaults to false.

equals Boolean Operator Examples

Element	Result
<code><equals value1="True" value2="true"/></code>	true
<code><equals value1="True" value2="true" exact="true"/></code>	false
<code><equals value1="apple" value2="apple" exact="true"/></code>	true
<code><equals value1="apple" value2="orange"/></code>	false
<code><equals value1=":[var1]" value2=":[var2]"/></code>	true if the value of var1 is equal to the value of var2.

matches Boolean Operator

A boolean operator used to determine if a particular value matches a pattern. It contains attributes "value", "pattern" and "exact", and has no child elements. It evaluates to true if and only if value of the "value" attribute matches the glob-style pattern contained in the value of the "pattern" attribute. If "exact" is true, then they must be case-sensitive match, otherwise they must be case-insensitive match.

matches Boolean Operator Attributes

Name	Type	Required	Configurable	Description
value	string	Yes	Yes	The value to be matched against the pattern.
pattern	string	Yes	Yes	The pattern that should be matched.
exact	boolean	No	No	True if a case-sensitive match should be performed, false otherwise. Defaults to false.

matches Boolean Operator Examples

Element	Result
<matches value="True" pattern="true"/>	true
<matches value="True" pattern="t*"/>	true
<matches value="blue" pattern="*u"/>	false
<matches value="True" pattern="t?ue"/>	true
<matches value="Tue" pattern="t?ue"/>	false
<matches value="True" pattern="t*" exact="true"/>	false
<matches value=":[var1]" pattern=":[var2]"/>	true if the value of var1 matches the pattern of var2.

not Boolean Operator

A boolean operator which negates the result of another boolean operator. It contains no attributes, and a single boolean operator child element. It evaluates to true if and only if the value of its contained operator is not true.

not Boolean Operator Examples

Element	Result
<not><istrue value="True"/></not>	false

<code><not><equals value1="apple" value2="orange"/></not></code>	true.
--	-------

and Boolean Operator

A boolean operator which logically ands results of other boolean operators. It contains no attributes. It may contain any number of boolean operator child elements. It evaluates to true if and only if all of its child elements evaluates to true.

and Boolean Operator Examples

Element	Result
<code><and/></code>	true
<code><and><istru value="True"/></and></code>	true
<code><and><equals value1="apple" value2="orange"/></and></code>	false
<code><and> <matches value="apple" value2="ap*e"/> <istru value="TRUE"/> <not><equals value1="apple" value2="orange"/></not></and></code>	true
<code><and> <matches value="apple" value2="ap*e"/> <istru value="TRUE"/> <equals value1="apple" value2="orange"/></and></code>	false

or Boolean Operator

A boolean operator which logically ors results of other boolean operators. It contains no attributes. It may contain any number of boolean operator child elements. It evaluates to true if and only if it contains at least one child element that evaluates to true.

or Boolean Operator Examples

Element	Result
<code><or/></code>	false

Element	Result
<or><istru e value="True"/></or>	true
<or><equals value1="apple" value2="orange"/></or>	false
<or> <matches value="apple" value2="p*e"/> <istru e value="FALSE"/> <equals value1="apple" value2="orange"/></or>	false
<or> <matches value="apple" value2="p*e"/> <not><istru e value="FALSE"/></not> <equals value1="apple" value2="orange"/></or>	true

Attribute Types

entityName Type

Attributes of type entityName have a maximum length of 512 and match the following pattern:

```
[a-zA-Z_] [0-9a-zA-Z-_. ] *
```

systemName Type

Attributes of type systemName have a maximum length of 64 and match the following pattern:

```
[a-zA-Z_] [0-9a-zA-Z-_. +] *
```

identifier Type

Attributes of type identifier have a maximum length of 32 and match the following pattern:

```
[a-zA-Z_] [0-9a-zA-Z_] *
```

pathName Type

Attributes of type pathName have a maximum length of 512 and match one of the following patterns:

```
"/" ( "/" identifier ) +
```

pathReference Type

Attributes of type pathReference have the following syntax:

```
pathReference:
    absolutePath
    relativePath

absolutePath:
    "/"
    "/" relativePath

relativePath:
    relativePathStart
    relativePathStart "/" relativePath

relativePathStart:
    ".."
    identifier
```

modifierEnum Type

Attributes of type modifierEnum have either ABSTRACT or FINAL as their value. In general, ABSTRACT indicates that the associated entity must be overridden by a derived component, and FINAL indicates that the associated entity may not be overridden.

accessEnum Type

Attributes of type accessEnum have either PUBLIC, PROTECTED, PATH or PRIVATE as their value. In general, PUBLIC indicates that the accessibility of the associated entity is unlimited, PROTECTED means that accessibility is limited to derived components and entities in the same path, PATH means that accessibility is limited to entities in the same path, and PRIVATE means that accessibility is limited to just the declaring component.

version Type

Attributes of type version match the following pattern:

```
[0-9]+\.[0-9]+
```

schemaVersion Type

Attributes of type schemaVersion currently may only have one value:

```
4.1
```

Component Change Compatibility

This section enumerates the changes that can be made to a component, and whether they are call and/or install compatible.

Component Attribute Changes

Nature of Change	Install Compatible	Call Compatible
non-final to final	no	yes
final to non-final	yes	yes
non-abstract to abstract	no	yes
abstract to non-abstract	yes	yes
more restrictive access	no	no
less restrictive access	yes	yes
change value of description, label, softwareVendor, or author	no	yes
change value of name or path	no	yes
change from simple to composite component	no	no
change from composite to simple component	no	no

Platform Changes

This section enumerates the changes that can be made to a platform, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
more general platform	yes	yes
more specific platform	no	yes
unrelated platform	no	yes

Here, a platform X is more specific than Y if X is a descendant of Y. It is more general if X is an ancestor of Y.

LimitToHostSet Changes

This section enumerates the changes that can be made to LimitToHostSet, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
any change to limitToHostSet	no	yes

Unlike platforms, the limitToHostSet names a generic, client-specified host set over which there is no explicit control. Its membership can change at any time, so there's no way to enforce notions of more or less specific host sets.

Extends Changes

This section enumerates the changes that can be made to Extends, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
new base component instance of original	no	yes
original base component instance of new	no	no
new base component unrelated to original	no	no
new base component install compatible with original	yes	yes
new base component call compatible with original	no	yes

Variable Changes

This section enumerates the changes that can be made to a variable, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
add a new variable	no	yes

Nature of Change	Install Compatible	Call Compatible
remove/rename a non-private variable	no	no
remove/rename a private variable	yes	yes
change final var default value	no	yes
change non-final var default value	yes	yes
non-final to final	no	yes
final to non-final	yes	yes
non-abstract to abstract	no	yes
abstract to non-abstract	yes	yes
more restrictive access	no	no
less restrictive access	no	yes

Component Ref List Changes

This section enumerates the changes that can be made to the Component Ref List, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
non-final to final	no	yes
final to non-final	yes	yes
new type instance of original	no	yes
original type instance of new	no	no
new type unrelated to original	no	no
new type install compatible with original	yes	yes
new type call compatible with original	no	yes

Component Ref Changes

This section enumerates the changes that can be made to Component Ref, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
non-final to final	no	yes
final to non-final	yes	yes
non-abstract to abstract	no	yes
abstract to non-abstract	yes	yes
change installMode	no	no
add a new component ref	no	yes
remove/rename a nested component ref	no	no
remove/rename a toplevel component ref	no	no
add/modify/remove nested component argList args	no	yes
add/modify/remove toplevel component argList args	yes	yes
new type instance of original	no	yes
original type instance of new	no	no
new type unrelated to original	no	no
new type install compatible with original	yes	yes
new type call compatible with original	no	yes
new nested comp instance of original	no	yes
original nested comp instance of new	no	no
new nested comp unrelated to original	no	no
new nested comp install compatible with original	yes	yes

Nature of Change	Install Compatible	Call Compatible
new nested comp call compatible with original	no	yes

Resource Changes

This section enumerates the changes that can be made to a resource, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
non-final to final	no	yes
final to non-final	yes	yes
non-abstract to abstract	no	yes
abstract to non-abstract	yes	yes
modify install path, name, group or user	no	yes
modify rsrcName, rsrcVersion	no	yes

Install, Control and Uninstall Block Changes

This section enumerates the changes that can be made to an install, control, or uninstall block, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
non-final to final	no	yes
final to non-final	yes	yes
non-abstract to abstract	no	yes
abstract to non-abstract	yes	yes
more restrictive access	no	no
less restrictive access	no	yes
add a new non-private block	no	yes
add a new private block	yes	yes
remove/rename a non-private block	no	no

Nature of Change	Install Compatible	Call Compatible
remove/rename a private block	yes	yes
change body of block	yes	yes
add/modify/remove local vars of block	yes	yes
add/modify/remove params of private block	yes	yes
add required param to non-private block	no	no
add optional param	yes	yes
remove optional/required param	yes	yes
rename optional param	yes	yes
rename required param of non-private block	no	no
change param from optional to required in non-private block	no	no
change param from required to optional	yes	yes
change param displayMode	yes	yes
change param prompt	yes	yes

Snapshot Block Changes

Snapshot blocks generally have the same compatibility matrix as other blocks listed above, except for dealing with their prepare, capture and cleanup blocks.

Nature of Change	Install Compatible	Call Compatible
add/modify/remove prepare/cleanup block	yes	yes
add/modify/remove capture step	yes	yes

Diff Ignore Changes

This section enumerates the changes that can be made to Diff Ignore, and whether they are call and/or install compatible.

Nature of Change	Install Compatible	Call Compatible
add/modify/remove an ignore directive	yes	yes

The ignore directives are only considered at diff runtime, and do not affect the state of existing snapshots in any way.

Attribute and Child Element Ordering

Within this document, the attributes and child elements that may be used within a particular element are typically documented using tables, one table listing attributes and another listing child elements. The order that the attributes are listed within the tables is not significant; they may appear in any order when used within the associated element. However, unless otherwise noted, the order that child elements are listed within the tables is significant; when used within the associated element, child elements must appear in the same relative order as they appear within the tables.

The N1 Service Provisioning System Software Component Schema

This chapter describes the XML schema for the N1 Service Provisioning System software components.

For an overview of the provisioning software's XML schema architecture, see Chapter 1.

Components

In the following tables, the “Config” column indicates whether or not a specific attribute may include substitution variable references in the form “[varName]”. For details on variable substitution, refer to Chapter 16.

The component Element

The entire component is enclosed by the <component> element. All versions of a component must have the same name and path. Configurable attributes of this element may reference component substitution variables.

component Element attributes

Name	Type	Required	Configurable	Description
xmlns	String	Yes	No	Required value: http://www.sun.com/schema/SPS
xmlns:xsi	String	Yes	No	Required value: http://www.w3.org/2001/ XML Schema-instance
xsi:schema Location	String	No	No	Recommended value: http://www.sun.com/schema/SPS component.xsd
access	One of PUBLIC PATH	No	No	The access mode of this component, as described below. Defaults to PUBLIC.
modifier	modifierEnum	No	No	The modifier of this component, as described below.
name	entityName	Yes	No	The name of the component
path	pathName	No	No	The absolute path of the component. If unspecified, the root path “/” is used by default.
description	String	No	No	The description of the component
label	String	No	No	A brief description of the component
software Vendor	String	No	No	The vendor of the software application modeled by this component
author	String	No	No	The author of this component
version	schemaVersion	Yes	No	The version of the component schema being used. The only value currently permitted is 4.0.
platform	String	No	No	The name of the host set containing the hosts that are considered valid physical targets for this component to be installed on, as described below.
limitToHostSet	String	No	No	The name of the host set containing the hosts that are considered valid targets for this component, as described below.

Name	Type	Required	Configurable	Description
installPath	String	Yes 1	Yes	The path used when this component is installed. For simple components, this value also serves as the root directory in which to install resources. Note: When an instance of this component is installed, the path is stored in universal format.

1 Only permitted in non-derived components.

component access attribute

The "access" attribute of the <component> element specifies the accessibility of the component, and is limited to values PATH and PUBLIC. It defaults to PUBLIC.

If access is PATH, then the component may only be referenced by other components in the same path as this component. It may not be direct installed by the client and may only be included using a nested reference in other components. If access is PUBLIC, no such restrictions apply.

component modifier attribute

The "modifier" attribute of the <component> element specifies override requirements for the component.

If ABSTRACT, then the component may not be installed. It serves only as a base component for other components to extend. Only abstract components may declare abstract child elements.

If FINAL, then the component may not be extended by another component.

If omitted, then the component may be extended and installed.

component platform attribute

The "platform" attribute of the <component> element specifies the name of the host set containing the hosts that are considered valid physical targets for this component to be installed on. If unspecified, any host that contains an RA residing on a supported platform is considered a to be a valid physical target. Otherwise, the physical targets for any plan that installs this component must be a subset of the hosts contained in the named host set. If the physical targets include a host not contained in the named host set, it is a plan run time error. It is also a plan run time error to specify a name that does not correspond to an existing host set. The plan run time errors described here are reported as preflight errors.

component limitToHostSet attribute

The "limitToHostSet" attribute of the <component> element specifies the name of the host set containing the hosts that are considered valid targets for this plan. If unspecified, all hosts are considered valid targets. Otherwise, the targets specified by the client must be a subset of the hosts contained in the named host set. If the targets include a host not contained in the named host set, it is a plan run time error. It is also a plan run time error to specify a name that does not correspond to an existing host set. The plan run time errors described here are reported as preflight errors.

There are two main differences between the platform and the limitToHostSet attributes. First, the platform names one of the predefined platform host sets, whereas the limitToHostSet names a user-defined host set. Accordingly, if the client wants to limit installation based on a custom host set, they should use "limitToHostSet". Secondly, when a component is targeted at a virtual host, the limitToHostSet is tested against the virtual host, whereas the platform is tested against the root physical host of that virtual host. Accordingly, setting a limitToHostSet but no platform allows a component to be installed on a particular set of virtual hosts that might reside on different physical platforms (as is the case with WebLogic applications). Setting a platform but no limitToHostSet allows a component to be installed on any host that is rooted by a physical host with the given platform. Setting both allows one to constrain both degrees.

component attribute inheritance

Except for installPath and limitToHostSet, component attributes are not inherited.

The installPath attribute is inherited, and may not be overridden by derived components. Note, however, that the base component may use component variables in specifying its value, and the value of these variables may be overridden.

The limitToHostSet attribute is also inherited, and may be overridden by derived components only if the base component did not specify a limitToHostSet. Note that since the limitToHostSet value names a mutable, client managed entity, the relation of host set can not be reasoned about in the same way as platforms.

The platform attribute is not inherited. However, the platform value of a derived component may be no more general than that of the base component. If the platform is unspecified in a derived component, then it also must be unspecified (or specified as "any") in the base component.

component Child Elements

Name	How Many?	Description
extends	0 or 1	The base component from which this component is derived.
varList	0 or 1	List of substitution variables used by this component and the configuration resources it contains.
resourceRef	0 or 1	Resources managed by this simple component. May not be used with <componentRefList>.
component RefList	0 or 1	List of components referenced by this composite component. May not be used with <resourceRef>.
installList	0 or 1 1	List of install step blocks, each providing a different way to install this component.
uninstallList	0 or 1 1	List of uninstall step blocks, each providing a different way to uninstall this component.
snapshotList	0 or 1	List of snapshot blocks, each providing a different way to capture the installed state of this component.
controlList	0 or 1	List of controls that are available for this component.
diff	0 or 1	List of directives used by the difference engine when running comparisons on components derived from this component.

1 Required in non-derived components.

The extends Element

The <extends> element is a child of the <component> element and is used to declare base component from which this component is derived. The base component may not be final.

This component will automatically inherit various attributes and elements of the base component. It can selectively override certain aspects of the inherited data. Allowances for inheritance and overrides are specified in the description of the attribute/element in question.

A component is said to be an instance of the component that it extends. It is also an instance of the components that the base component is an instance of.

extends Child elements

Name	How Many?	Description
type	1	Specifies the base component.

The type Element

The <type> element is a child of the <extends>, <componentRefList> and <componentRef> elements and names a base component type.

type Element attributes

Name	Type	Required	Configurable	Description
name	systemName	Yes	No	The name of the system type component that serves as the base type.

The component varList Element

The component <varList> element is a child of the <component> element and is used to declare the list of component-scoped substitution variables used by this component and the configuration resources it contains.

component varList Child elements

Name	How Many?	Description
var	1 or more	A component substitution variable declaration, including name and default value.

component varList Inheritance and Overriding

By default, a derived component inherits the accessible <varList> element contents of its base component. When a derived component declares a <varList>, its contents are effectively merged with those of the base component. The derived component may declare new <var> elements and override inherited ones, but may not remove elements declared by the base component.

The component var Element

The component <var> element is a child of the component <varList> element and is used to declare a component substitution variable including name and default value.

component var Element attributes

Name	Type	Required	Configurable	Description
access	access Enum	No	No	The access mode of the variable, as described below. Default is PUBLIC.
modifier	modifier Enum	No	No	The modifier of the variable, as described below.
name	identifier	Yes	No	The name of the substitution variable. The name must be unique among every other substitution variable in the enclosing <varList>.

Name	Type	Required	Configurable	Description
default	String	Yes 1	Yes	The default value of the substitution variable, which may include references to other substitution variables, target host attributes, and installed component variables.

1 May not be included in abstract variables.

var access Attribute

The "access" attribute of the <var> element specifies the accessibility of the variable.

If PUBLIC, access is not restricted in any way.

If PROTECTED, access is limited to derived components and entities in the same path.

If PATH, access is limited to entities in the same path.

If PRIVATE, access is limited to this component.

var modifier Attribute

The "modifier" attribute of the <var> element specifies the override requirements for the variable.

If ABSTRACT, the variable "default" attribute is omitted and must be specified by non-abstract derived components. Variables may only be declared abstract if the component is also declared abstract. Abstract variables may not be private. Non-abstract variables must declare a default value.

If FINAL, the variable may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the variable.

var Inheritance and Overriding

By default, a derived component inherits all accessible variables of its base component, including access mode, modifier, and default value.

A derived component may define additional variables by using names that are not among the variables inherited from the base component. A derived component may override the default value, modifier, and access mode of a non-final inherited variable

by re-declaring a variable with the same name. When a variable is overridden, the entire content of the variable must be re-declared, including default value, access mode, and modifier. The default value is specified if and only if the overriding variable is non-abstract. The access mode may be no more restrictive than the base component mode.

When a variable is overridden, all references to the variable evaluate to the overridden value, including those in the base component.

Note – If the derived component is declared as non-abstract, then any abstract variables declared by the base component must be overridden by the derived component.

The resourceRef Element

The <resourceRef> element is a child of the <component> element and specifies the resource managed by this component. It may not be used in conjunction with the <componentRefList> element. Configurable attributes of this element and its children may reference component substitution variables.

Note – Resources have implicit PUBLIC access mode.

A component is considered a simple component if it is derived from a simple component, or is a non-derived component that contains a <resourceRef> element. Derived components may only contain a <resourceRef> element if they are derived from a simple component.

resourceRef Element attributes

Name	Type	Req,	Configurable	Description
modifier	modifierEnum	No	No	The modifier of the resource, as described below.

resourceRef modifier Attribute

The "modifier" attribute of the <resourceRef> element specifies the override requirements for the resource.

If ABSTRACT, the resourceRef <resource> element is omitted and must be specified by non-abstract derived components. ResourceRefs may only be declared abstract if the component is also declared abstract. Non-abstract resourceRefs must declare a <resource> element.

If FINAL, the resourceRef may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the resourceRef.

resourceRef Child elements

Name	How Many	Description
installSpec	1 1	Specifies how the resource is to be installed.
resource	1 2	Identifies the associated resource.

1 Only permitted in non-derived components. 2 May not be included in abstract resources.

resourceRef Inheritance and Overriding

By default, a derived component inherits the <resourceRef> element of its base component.

A derived component may override the modifier and <resource> element of a non-final inherited <resourceRef> element by re-declaring the <resourceRef> element. When a <resourceRef> element is overridden, the <installSpec> element is omitted, as its contents may not be overridden. The <resource> element is specified if and only if the overriding <resourceRef> is non-abstract.

When a <resourceRef> is overridden, then all uses of the resource (including <deployResource> and <addResource>) will resolve to the overridden value, including uses within the base component.

Note – If the derived component is declared as non-abstract, and the <resourceRef> element of the base component is abstract, then the derived component must override the <resourceRef> element.

The installSpec Element

The <installSpec> element is a child of the <resourceRef> element and is used to specify the way in which the associated resource is to be installed.

This element is inherited by derived components, and may not be overridden. Note, however, that the base component may use component variables in specifying values for <installSpec> attributes, and the values of these variables may be overridden.

installSpec Element attributes

Name	Type	Required	Configurable	Description
name	String	Yes	Yes	The name to use for the resource when it is installed.
path	String	No	Yes	The directory path in which to install the resource. Relative directories are considered relative to the installPath attribute of the containing component. If unspecified, the component installPath attribute is used by default.

Name	Type	Required	Configurable	Description
permissions	String	No	Yes	The permissions to assign to this resource when installed, in the format of an octal triplet, as defined by the UNIX chmod command. The first digit represents permissions for the owner of the file, the second is permissions for other users in the file's group, and the third is permissions for other users. Each digit is the sum of the numbers corresponding to the desired permissions where 1 is execute, 2 is write and 4 is read. So "777" gives everyone read, write and execute permission on the file. If unspecified, the resource is installed with default permissions.
user	String	No	Yes	The user to assign as the owner of this resource when installed. If unspecified, the user to use is determined by the plan executor.
group	String	No	Yes	The group to assign to this resource when installed. If unspecified, the group to use is determined by the plan executor.
deploy Mode	One of: ADD_TO or REPLACE	No	Yes	Specifies the way in which the associated directory resource is deployed. ADD_TO means directory contents will be added to any existing files in the target directory. REPLACE means directory contents will replace all existing files in the target directory. If unspecified, REPLACE is used by default. This attribute is ignored for non-directory resources.
diffDeploy	Boolean	No	Yes	Specifies whether the resource should be deployed in differential deploy mode. If unspecified, false is used by default. If differential deploy mode is enabled, then only resources which have not previously been deployed will be deployed.

The resource Element

The <resource> element is a child of the <resourceRef> element and is used to identify the resource to be deployed by this component.

If the referenced resource is a configurable resource, it may contain substitution variable references to any component-scoped variable accessible to the containing component.

resource Element attributes

Name	Type	Required	Configurable	Description
name	String	Yes	No	The full path name of a resource managed by the resource manager.
version	version	Yes	No	The version of the resource managed by the resource manager.

The componentRefList Element

The <componentRefList> element is a child of the <component> element and specifies the list of components referenced by this component. It may not be used in conjunction with the <resourceRef> element. Configurable attributes of this element and its children may reference component substitution variables.

A component is considered a composite component if it is derived from a composite component, or is a non-derived component that does not contain a <resourceRef> element. Derived components may only contain a <componentRefList> element if they are derived from a composite component.

componentRefList Element attributes

Name	Type	Req.	Configurable	Description
modifier	FINAL	No	No	The modifier of the list, as described below.

componentRefList modifier Attribute

The "modifier" attribute of the <componentRefList> element specifies the override requirements for the list. If specified, the value must be FINAL.

If FINAL, derived components may not declare new <componentRef> elements.

If unspecified, then derived components can add new <componentRef> elements.

In either case, derived components may override the <type> element, and non-final inherited <componentRef> elements. If the base component's componentRefList modifier is final, then that of the derived component must also be final.

Note – A final modifier on the <componentRefList> element does not imply that each of the contained <componentRef>s are also final.

componentRefList Child elements

Name	How Many?	Description
type	0 or 1	Specifies the type that all referenced components must be instances of. If unspecified, referenced components may be of any type.
componentRef	0 or more	A component reference.

componentRefList Inheritance and Overriding

By default, a derived component inherits the <componentRefList> element contents of its base component. When a derived component declares a <componentRefList>, its contents are effectively merged with those of the base component. The derived component may declare new <componentRef> elements and override inherited ones, but may not remove elements declared by the base component.

A derived component may override the <type> element declared by the <componentRefList> of the parent component by re-declaring the <type> within its <componentRefList>. In this case, the overridden type must be an instance of the original type, or the original must be unspecified. Furthermore, all referenced components must be instances of the overridden type, including those inherited from the base component.

The componentRef Element

The <componentRef> element is a child of the <componentRefList> element and specifies a component referenced by this component.

Note – Component refs have implicit PUBLIC access mode.

The componentRef Element attributes

Name	Type	Required	Configurable	Description
modifier	modifierEnum	No	No	The modifier of the componentRef, as described below.
name	identifier	Yes	No	A local name for the referenced component. Must be unique among all sibling <componentRef> elements.
installMode	One of: ■ TOPLEVEL ■ NESTED	No	No	The way in which the referenced component should be installed. If unspecified, NESTED is used by default.

componentRef modifier Attribute

The "modifier" attribute of the <componentRef> element specifies the override requirements for the component reference.

If ABSTRACT, the component reference <component> element is omitted and must be specified by non-abstract derived components. Component references may only be declared abstract if the component is also declared abstract. Non-abstract component references must declare a <component> element.

If FINAL, the component reference may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the component reference.

componentRef installMode Attribute

The installMode attribute of the <componentRef> element specifies the way in which the referenced component should be installed and targeted thereafter. If the referenced component is installed as TOPLEVEL, then it can be used by any other component just as if it had been directly installed by a plan. However, if the referenced component is installed as NESTED, then its installation is implicitly scoped to that of the referencing component, and its services are only available to the referencing component.

Logically, a nested referenced component defines a finer-grained unit of functionality required by the referencing component, but not otherwise useful to other components. Whereas a toplevel referenced component defines services that will be used by the referencing component, but may also be used by other components.

The lifetime of a nested referenced component is implicitly scoped to that of the referencing component. The nested referenced component can only be installed during the installation of the referencing component, and is implicitly uninstalled when the referencing component is uninstalled. In contrast, the lifetime of a toplevel referenced component is not tied to that of the referencing component in any way. The referencing component may install a toplevel referenced component when it is installed, or the referenced component have been installed previously or later by other means. When the referencing component is uninstalled, a toplevel referenced component will remain installed unless it is explicitly uninstalled by the referencing component. Other components are also free to uninstall it.

The componentRef Child elements

Name	How Many?	Description
type	0 or 1	Specifies the type that the referenced component must be an instance of. It must be an instance of the type specified by the enclosing <componentRefList>. If unspecified, the type specified by the enclosing <componentRefList> is used.
argList	0 or 1	A list of values to be used as component variable settings for the referenced component when it is installed.
component	1 1	The referenced component.

1 May not be included in abstract componentRefs.

componentRef Inheritance and Overriding

By default, a derived component inherits all component references of its base component.

If the <componentRefList> element of the base component is non-final, the a derived component may define additional component references by using names that are not among those of the component references inherited from the base component.

A derived component may override a component reference of a non-final inherited component reference by redeclaring a component reference with the same name. When a component reference is overridden, the entire content of the component reference must be re-declared. The overriding installMode must be the same as that of the original reference. The overriding <type> element must be an instance of the original type. The overriding <argList> element is merged with the original, as described in the following section. The <component> element is specified if and only if the overriding reference is non-abstract.

When a component reference is overridden, all uses of the component reference evaluate to the overridden value, including those in the base component.

Note – If the derived component is declared as non-abstract, then any abstract component references declared by the base component must be overridden by the derived component.

The componentRef/argList Element

The <argList> child element of the <componentRef> element specifies a list of values to be used as component variable settings for the referenced component when it is installed. The format of the <argList> is the same as that of the <argList> child element of the <call> step. Each attribute of the <argList> element names a component variable in the referenced component or the declared type, if the reference is ABSTRACT. The value of the <argList> attribute is the override value that should be used for the named component variable when the referenced component is installed.

Note – When a component reference is overridden by a derived component, the <argList> of the base and derived components are effectively merged by first applying the contents of the <argList> of the base component to the referenced component, and then applying the <argList> of the derived component. When processing the <argList> of the base component reference, only variables defined in the declared type of the base component reference are considered.

Component variables not named in the <argList> will use their default value when installed. If no <argList> is specified, then the referenced component will be installed using default values for all its variables. The variables of the referenced component named in the <argList> must be accessible to the referencing component. Furthermore, they must be declared with access mode PUBLIC or PROTECTED, and may not be FINAL.

Note – For top-level referenced components, the <argList> element is only used if the referenced component is installed by the referencing component. The referenced component may have been installed by other means, in which case the <argList> has no bearing.

The componentRef/component Element

The <component> child element of the <componentRef> element identifies the referenced component. It has the same structure as the <component> repository component targeter, except that the "host" attribute is not permitted. The referenced component version must exist in the repository at the time the containing component is saved.

Note – If the "version" attribute is not specified, then the version is resolved to the latest version of the referenced component that exists at the time the containing component is saved. It is a save time error if no versions of the referenced component exist. Once the containing component is saved, the versions of all the components it references are locked, and cannot be modified without creating a new version of the containing component.

The installList Element

The <installList> element is a child of the <component> element and contains one or more named install step blocks, each providing a different way to install this component. Many components will have only one install block as a child of this element.

Multiple install blocks can be used when different steps are required for different installation environments. For example, steps to deploy an EJB app to a server cluster versus deploy to a single managed server, or steps to install for the first time versus steps to upgrade the application.

installList Child elements

Name	How Many?	Description
installSteps	1 or more	A named install block containing steps that may be executed to install this component.

installList Inheritance and Overriding

By default, a derived component inherits the accessible <installList> element contents of its base component. When a derived component declares an <installList>, its contents are effectively merged with those of the base component. The derived component may declare new <installSteps> elements and override inherited ones, but may not remove elements declared by the base component.

The installSteps Element

The <installSteps> element is a child of the <installList> element and lists the sequence of steps to be executed in order to install this component. When an <install> step causes this component to be installed, the steps listed here are executed in order. Typically, the install steps of a simple component include a <deployResource> step, and the install steps of a composite component include one or more <install> steps to install referenced components.

installSteps Element attributes

Name	Type	Required	Configurable	Description
access	accessEnum	No	No	The access mode of the install block, as described later. Default PUBLIC.
modifier	ModifierEnum	No	No	The modifier of the install block, as described later.
name	entityName	Yes	No	The name of the install block. The name must be unique among all install blocks in the containing <installList>.
description	String	No	No	A description of the install block. Useful for documentation purposes.

installSteps access Attribute

The "access" attribute of the <installSteps> element specifies the accessibility of the install block.

If PUBLIC, access is not restricted in any way.

If PROTECTED, access is limited to derived components and entities in the same path.

If PATH, access is limited to entities in the same path.

If PRIVATE, access is limited to this component.

Note – Only PUBLIC blocks may be direct run.

installSteps modifier Attribute

The "modifier" attribute of the <installSteps> element specifies the override requirements for the install block.

If ABSTRACT, the block may not include a body. The body must be specified by non-abstract derived components. Install blocks may only be declared abstract if the component is also declared abstract. Abstract blocks may not be private. Non-abstract blocks must declare a body.

If FINAL, the install block may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the block.

installSteps Child elements

The <installSteps> element children consist of an optional <paramList> element followed by the body, where the body consists of an optional local <varList> element followed by zero or more "shared ", or "component install-only" steps. Note that the body is not included if the install block is declared abstract.

installSteps Inheritance and Overriding

By default, a derived component inherits all accessible install blocks of its base component.

A derived component may define additional install blocks by using names that are not among those of the install blocks inherited from the base component. A derived component may override a non-final inherited install block by redeclaring a block with the same name. Note that blocks are overridden using name alone, and may not be overloaded based on parameters. When a block is overridden, the entire content of the block must be re-declared, including the access mode, modifier, parameters and body. The body is specified if and only if the overriding block is non-abstract. The access mode may be no more restrictive than that of the base component.

The signature of the overriding block in the derived component must be compatible with that of the base component. By compatible, it is meant that any arguments that are acceptable to the base block must also be acceptable to the derived block.

A derived block is compatible with a base block if:

- It does not declare a new required parameter.
- It does not redefine a parameter that was optional in the parent block to be required.

The following signature changes are considered compatible:

- Removal of a parameter (required or optional).
- Making a required parameter optional.
- Addition of an optional parameter.

When a block is overridden, then all references to the block evaluate to the overridden value, including those in the base component.

Note – If the derived component is declared as non-abstract, then any abstract blocks declared by the base component must be overridden by the derived component.

A block in a derived component may explicitly call into a block defined by the base component even if the derived overrides the block using the `<superComponent>` targeter.

The paramList Element

The `<paramList>` element is a child of the `<installSteps>`, `<uninstallSteps>`, `<snapshot>` and `<control>` elements and is used to declare a list of parameters that may be used by the steps of the enclosing element. The value of the parameters are defined by the caller based on the content of the caller's `<argList>` element. For example, in the case of a `<paramList>` within an `<installSteps>` block, parameter values are defined based on the `<argList>` of the `<install>` step that invoked the `<installSteps>` block.

The steps of the enclosing element may use locally-scoped variables declared in the local `<varList>` element, parameters declared in the `<paramList>` element, and component-scoped variables declared in the component `<varList>` element of the enclosing component. If a `<paramList>` parameter has the same name as a component `<varList>` variable, the value of the parameter is used. In this case, the parameter is said to "hide" the component variable. Hiding is not permitted between local variables and parameters, as their names must be distinct.

paramList Child elements

Name	How Many?	Description
param	1 or more	A parameter declaration, including name and default value.

The param Element

The <param> element is a child of the <paramList> element and is used to declare a parameter including name and default value. The default value is used only if the caller does not explicitly pass a value for this parameter. If the default value is unspecified and the caller does not explicitly pass a value for this parameter, a preflight error is raised at plan runtime.

The param element includes optional prompt and displayMode attributes for use when the containing install/uninstall/control block is invoked directly by the client rather than from a plan or another component.

param Element attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of the parameter. The name must be unique among every other local variable and parameter declared by the enclosing element.
prompt	String	No	No	The UI text to display when prompting for the value of the parameter. If unspecified, name is used by default.
default	String	No	Yes	The default value of the parameter, which may include references to component variables, target host attributes, and installed component variables, but not other parameters.

Name	Type	Required	Configurable	Description
display Mode	One of: PASSWORD CLEAR BOOLEAN	No	No	The display mode of the parameter. If unspecified, CLEAR is used by default. If PASSWORD, the client-entered value will be hidden (e.g. not shown or replaced with ***). If BOOLEAN, the parameter is entered using a checkbox. Otherwise if CLEAR or BOOLEAN, it is safe for the value to be displayed as entered.

The local varList Element

The local <varList> element is a child of the <installSteps>, <uninstallSteps>, <snapshot> and <control> elements and is used to declare a list of variables that may be used by the steps of the enclosing element. The values of the variables are defined at the point of declaration, and cannot be redefined.

The steps of the enclosing element may use locally-scoped variables declared in the local <varList> element, parameters declared in the <paramList> element, and component-scoped variables declared in the component <varList> element of the enclosing component. If a local <varList> variable has the same name as a component <varList> variable, the value of the local variable is used. In this case, the local variable is said to "hide" the component variable. Hiding is not permitted between local variables and parameters, as their names must be distinct.

local varList Child elements

Name	How Many?	Description
var	1 or more	A local variable declaration, including name and default value.

The local var Element

The local <var> element is a child of the local <varList> element and is used to declare a local variable including name and value.

local var Element attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of the local variable. The name must be unique among every other local variable and parameter declared by the enclosing element.
default	String	Yes	Yes	The default value of the local variable, which may include references to other local variables declared earlier, parameters, component variables, target host attributes, and installed component variables.

uninstallList Element

The <uninstallList> element is a child of the <component> element and contains one or more named uninstall step blocks, each providing a different way to uninstall this component. Many components will have only one uninstall block as a child of this element. Multiple uninstall blocks can be used when different steps are required for different installation environments. For example, steps to undeploy an EBJ app from a server cluster versus undeploy from a single managed server. Uninstall blocks often correspond one to one with install blocks, and in such cases, by convention use the same name to indicate the correspondence.

uninstallList Child elements

Name	How Many?	Description
uninstallSteps	1 or more	A named uninstall block containing steps that may be executed to uninstall this component.

uninstallList Inheritance and Overriding

By default, a derived component inherits the accessible <uninstallList> element contents of its base component. When a derived component declares an <uninstallList>, its contents are effectively merged with those of the base component. The derived component may declare new <uninstallSteps> elements and override inherited ones, but may not remove elements declared by the base component.

The uninstallSteps Element

The <uninstallSteps> element is a child of the <uninstallList> element and lists the sequence of steps to be executed in order to uninstall this component. When an <uninstall> step causes this component to be uninstalled, the steps listed here are executed in order. Typically, the uninstall steps of a simple component include an <undeployResource> step, and the uninstall steps of a component component include one or more <uninstall> steps to uninstall referenced components.

uninstallSteps Element attributes

Name	Type	Required	Configurable	Description
access	accessEnum	No	No	The access mode of the uninstall block, as described below. Default PUBLIC.
modifier	modifierEnum	No	No	The modifier of the uninstall block, as described below.

Name	Type	Required	Configurable	Description
name	entityName	Yes	No	The name of the uninstall block. The name must be unique among all uninstall blocks in the containing <uninstallList>.
description	String	No	No	A description of the uninstall block. Useful for documentation purposes.

uninstallSteps access Attribute

The "access" attribute of the <uninstallSteps> element specifies the accessibility of the uninstall block.

If PUBLIC, access is not restricted in any way.

If PROTECTED, access is limited to derived components and entities in the same path.

If PATH, access is limited to entities in the same path.

If PRIVATE, access is limited to this component.

Note – Only PUBLIC blocks may be direct run.

uninstallSteps modifier Attribute

The "modifier" attribute of the <uninstallSteps> element specifies the override requirements for the uninstall block.

If ABSTRACT, the block may not include a body. The body must be specified by non-abstract derived components. Uninstall blocks may only be declared abstract if the component is also declared abstract. Abstract blocks may not be private. Non-abstract blocks must declare a body.

If FINAL, the uninstall block may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the block.

uninstallSteps Child elements

The <uninstallSteps> element children consist of an optional <paramList> element followed by a body, which consists of an optional local <varList> element followed by an optional <dependantCleanup> block followed by zero or more "shared " or "component uninstall-only" steps. Note that the body is not included if the uninstall block is declared abstract.

uninstallSteps Inheritance and Overriding

By default, a derived component inherits all accessible uninstall blocks of its base component. Semantics for overriding an uninstall block are the same as those for overriding an install block.

The dependantCleanup Block Element

The <dependantCleanup> element is a child of the <uninstallSteps> element, and specifies a set of steps to be executed in order to remove components that currently depend on the calling component. It has no attributes, and may include any number of steps that are permitted within the scope of the containing uninstall block. When included, it causes the check for dependant components to be deferred until after the contents of the block have been executed. If there are still dependant components after the block has been executed, then the uninstall fails and the component remains installed. If there are no dependant components remaining, then the uninstall proceeds with the remaining steps.

Note – If no <dependantCleanup> block is included in an uninstall block, then the block fails immediately if dependant components exist.

The <dependantCleanup> block is often used in conjunction with the <allDependants> targeter to bulk uninstall dependant components.

The snapshotList Element

The <snapshotList> element is a child of the <component> element and contains one or more named snapshot blocks, each providing a different way to capture the installed state of this component on the target host. Multiple snapshot blocks can be used to capture different aspects of the installed state, which allows for fine-grained differencing of the captured installed state and current state of the component.

snapshotList Child elements

Name	How Many?	Description
snapshot	1 or more	A named snapshot block that may be executed to capture the installed state of this component.

snapshotList Inheritance and Overriding

By default, a derived component inherits the accessible <snapshotList> element contents of its base component. When a derived component declares a <snapshotList>, its contents are effectively merged with those of the base component. The derived component may declare new <snapshot> elements and override inherited ones, but may not remove elements declared by the base component.

The snapshot Element

The <snapshot> element is a child of the <snapshotList> element and defines a sequence of steps to be executed in order to capture the installed state this component. When a <createSnapshot> or <addSnapshot> step names this snapshot block, the steps within the <prepare> block are executed in order, then files named with the <capture> block are captured within the capture area of the target machine, and finally steps within the <cleanup> block are executed in order.

A snapshot block is also used to compare the current state of a component against its state at the time of installation. In particular, the <prepare> steps are re-executed at the target machine, then files captured at install time are compared against the current state of the files, and the <cleanup> steps are re-executed.

snapshot Element attributes

Name	Type	Required	Configurable	Description
name	entityName	Yes	No	The name of the snapshot block. The name must be unique among all snapshot blocks in the containing <snapshotList>.
description	String	No	No	A description of the snapshot block. Useful for documentation purposes.

snapshot access Attribute

The "access" attribute of the <snapshot> element specifies the accessibility of the snapshot block.

If PUBLIC, access is not restricted in any way.

If PROTECTED, access is limited to derived components and entities in the same path.

If PATH, access is limited to entities in the same path.

If PRIVATE, access is limited to this component.

snapshot modifier Attribute

The "modifier" attribute of the <snapshot> element specifies the override requirements for the snapshot block.

If ABSTRACT, the block may not include a body. The body must be specified by non-abstract derived components. Snapshot blocks may only be declared abstract if the component is also declared abstract. Abstract blocks may not be private. Non-abstract blocks must declare a body.

If FINAL, the snapshot block may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the block.

snapshot Child elements

Name	How Many?	Description
paramList	0 or 1	A list of parameters for use within the prepare, capture and cleanup blocks of this snapshot.
varList	0 or 1	A list of local variables for use within the prepare, capture and cleanup blocks of this snapshot.
prepare	0 or 1	Contains steps to be executed in preparation for the file capture or comparison.
capture	0 or 1	Contains a list of files and directories to be captured as part of this snapshot
cleanup	0 or 1	Contains steps to be executed after the capture or comparison has completed.

Note – If this snapshot is to be called from a <createSnapshot> step, then it may not declare any required parameters in its <paramList> element.

The varList, prepare, capture and cleanup elements collectively define the body of the snapshot. The body is not included if the snapshot block is declared abstract.

snapshot Inheritance and Overriding

By default, a derived component inherits all accessible snapshot blocks of its base component. Semantics for overriding a snapshot block are the same as those for overriding an install block.

Note – There is no way to call the base component snapshot block's prepare block from a derived component snapshot block's prepare block. Same goes for cleanup. To do this, the base component must factor its prepare/cleanup steps into a control block that may be called by the derived component.

The snapshot prepare Child element

The <prepare> element is a child of the <snapshot> element and defines a sequence of steps to be executed in preparation for the file capture or comparison. Note that these steps are executed both as a result of a <createSnapshot> or <addSnapshot> step that targets this snapshot, and a comparison run that targets this snapshot. In all cases, these steps are executed prior to capturing any files or performing any comparisons.

prepare Child elements

The <prepare> element children consist of one or more <call>, <execNative> and/or <transform> steps. No other steps are permitted. The contained steps may reference local parameters and variables declared by the snapshot block, as well as unhidden component substitution variables.

The snapshot capture Child element

The <capture> element is a child of the <snapshot> element and defines the files and resources that are to be captured as part of this snapshot. Note that the capture occurs only after the steps in the <prepare> block have executed, and that once the capture is complete, the steps in the <cleanup> block will execute.

capture Child elements

The <capture> element children consist of one or more <addFile>, <addSnapshot>, and/or <addResource>, elements. The file and directories listed in this block are captured in the order specified. The contained children may reference local parameters and variables declared by the snapshot block, as well as unhidden component substitution variables.

The capture addFile Child element

The <addFile> element is a child of the <capture> element and denotes a file that is to be captured as part of the containing snapshot.

addFile Element attributes

Note – Values can be multibyte character strings.

Name	Type	Required	Configurable	Description
path	String	Yes	Yes	The pathname of a file or directory on the filesystem of the target host.
ownership	One of: SET_SELF ADD_SELF ADD_TEMP	No	No	The ownership option for the captured file, as described below.
filter	One of: DIRECTORIES FILES BOTH	No	No	When the pathname refers to a directory, this attribute is used to indicate whether the directory itself, the files it contains, or both should be captured. If unspecified, BOTH is used by default. If the pathname is not a directory, this attribute is ignored and the file is captured directly.
recursive	Boolean	No	No	When the pathname refers to a directory, this attribute is used to indicate whether subdirectories should be recursively captured using the current filter settings. If unspecified, true is used by default. If the pathname is not a directory, this attribute is ignored and the file is captured directly.
displayName	String	No	Yes	Text to be included in the display when this snapshot entry is compared against another.

addFile ownership attribute

The “ownership” attribute of the <addFile> element specifies the ownership option for the captured file. If unspecified, SET_SELF is used by default.

One aspect of the installed state that is captured by the snapshot is file and directory ownership. This is not the same as UNIX permissions; it is more closely tied to the notion of reference counts. Specifically, a file or directory can be captured as being owned by one or more snapshots. If the file owner later changes as a result of another component installation, this can be recognized and reported when the file is compared against its initial state. This facilitates tracking down differences that result from one component unintentionally overwriting files associated with another component. Snapshot ownership information is captured in a repository on the target host known as the owners table.

The values of the ownership attribute have the following semantics:

- If the ownership value is SET_SELF, then the owners table should be updated so that it contains a single entry for the associated file or directory. That entry lists the executing installed component and snapshot as owners. It also lists the capture area ID of the captured contents of the file or directory.
- If the ownership value is ADD_SELF, then the installed component and snapshot should be added as additional owners, sharing the existing capture area ID as the previous owners.
- The ownership value ADD_TEMP is like ADD_SELF except that a new capture is always created and its ID is always used for the new entry rather than sharing the one of the other owners.

The capture addSnapshot Child element

The <addSnapshot> element is a child of the <capture> element and denotes that an external snapshot block should be executed and its contents added to this snapshot.

Using an <addSnapshot> step is semantically equivalent to adding the prepare steps of the called snapshot to the end of the calling snapshot's prepare block, adding the cleanup steps of the called snapshot to the start of the calling snapshot's cleanup block, and adding the capture steps of the called snapshot to the calling snapshot's capture block in place of the <addSnapshot> step. Accordingly, care should be taken that the files referenced by the called and calling snapshot blocks do not conflict.

Any number of <addSnapshot> steps may appear within a <capture> element, and the called snapshot may itself contain any number of <addSnapshot> steps within its <capture> element.

Note – When files are added to a snapshot indirectly using <addSnapshot> callouts, the topmost component that initiated the snapshot capture is considered the owner of the files, as opposed to the component that contained the <addFile> directive. Similarly, when a diff is performed on a snapshot, only diff ignore directives of the topmost component that initiated the snapshot are considered. Diff ignore directives contained on components visited as a result of <addSnapshot> callouts are not considered.

addSnapshot Element attributes

Name	Type	Required	Configurable	Description
blockName	String	Yes	No	The name of the external snapshot block to execute.

addSnapshot Child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the snapshot block.
installed component targeter	0 or 1	Identifies the component containing the snapshot block. If unspecified, <thisComponent> is used.

The capture addResource Child element

The <addResource> element is a child of the <capture> element and denotes a resource associated with the component that is to be captured as part of the containing snapshot. This element may only be included in simple components.

This element serves as a syntactic shorthand for an equivalent <addFile> element, as follows:

- If the associated resource is a directory resource with deployMode ADD_TO, then <addResource> is equivalent to: <addFile path="path of deployed directory" filter="FILES" displayName=" resourceSourcePath"/>

- If the associated resource is a directory resource with deployMode REPLACE, then <addResource> is equivalent to: <addFile path="path of deployed directory" displayName="resourceSourcePath"/>
- Otherwise, the associated resource is a file-based resource and <addResource> is equivalent to: <addFile path="path of deployed file" displayName="resourceSourcePath"/>

The snapshot cleanup Child element

The <cleanup> element is a child of the <snapshot> element and defines a sequence of steps to be executed after a file capture or comparison has completed. Note that these steps are executed both as a result of a <createSnapshot> or <addSnapshot> step that targets this snapshot, and a comparison run that targets this snapshot. In all cases, these steps are executed after capturing all files or performing comparisons. Cleanup blocks are generally used to remove any temporary files created by the prepare block.

The snapshot cleanup Child elements

The <cleanup> element children consist of one or more <call>, <execNative> and/or <transform> steps. No other steps are permitted. The contained steps may reference local parameters and variables declared by the snapshot block, as well as unhidden component substitution variables.

The controlList Element

The <controlList> element is a child of the <component> element and lists the control blocks available for this component.

controlList Child elements

Name	How Many?	Description
------	-----------	-------------

control	1 or more	A control block.
---------	-----------	------------------

controlList Inheritance and Overriding

By default, a derived component inherits the accessible <controlList> element contents of its base component. When a derived component declares a <controlList>, its contents are effectively merged with those of the base component. The derived component may declare new <control> elements and override inherited ones, but may not remove elements declared by the base component.

The control Element

The <control> element is a child of the <controlList> element and defines a control block available for this component. A control block is a sequence of steps that may be performed once this component has been installed. For example, component for a database application might include control blocks to “startup” or “shutdown” the database. A <call> step can invoke a control block by name, which causes the control block steps to be executed in order.

control Element attributes

Name	Type	Required	Configurable	Description
access	accessEnum	No	No	The access mode of the control block, as described below. Default PUBLIC.
modifier	modifierEnum	No	No	The modifier of the control block, as described below.
name	entityName	Yes	No	The name of the control block. This is referenced from a <call> step to execute the control.
description	String	No	No	The description of the control block.

control access Attribute

The "access" attribute of the <control> element specifies the accessibility of the control block.

If PUBLIC, access is not restricted in any way.

If PROTECTED, access is limited to derived components and entities in the same path.

If PATH, access is limited to entities in the same path.

If PRIVATE, access is limited to this component.

Note – Only PUBLIC blocks may be direct run.

control modifier Attribute

The "modifier" attribute of the <control> element specifies the override requirements for the control block.

If ABSTRACT, the block may not include a body. The body must be specified by non-abstract derived components. Control blocks may only be declared abstract if the component is also declared abstract. Abstract blocks may not be private. Non-abstract blocks must declare a body.

If FINAL, the control block may not be overridden by derived components.

If unspecified, then derived components can choose whether or not to override the block.

control Child elements

The <control> element children consist of an optional <paramList> element followed by a body, which consists of an optional local <varList> element followed by zero or more "shared" steps. Note that the body is not included if the control block is declared abstract.

control Inheritance and Overriding

By default, a derived component inherits all accessible control blocks of its base component. Semantics for overriding a control block are the same as those for overriding an install block.

The diff Element

The <diff> element is a child of the <component> element and contains a list of directives used by the difference engine when running comparisons on components derived from this component.

diff Child elements

Name	How Many?	Description
ignore	1 or more	A directory path to ignore during comparison.

diff Inheritance and Overriding

A derived component automatically inherits all difference ignore directives declared by the base component. It may also declare additional ignore directives in its own <diff> element. Inherited directives may not be removed.

The ignore Element

The <ignore> element is a child of the <diff> element and specifies a filename path pattern to ignore when using this component in a comparison. This element is typically for files and directories created by the application once installed, such as log files. The configurable attributes of this element may reference component substitution variables.

ignore Element attributes

Name	Type	Required	Configurable	Description
------	------	----------	--------------	-------------

path	String	Yes	Yes	A glob-style pattern matching the filename paths to ignore. For example, <code>"/logs/*.log"</code>
------	--------	-----	-----	---

Installed Component Targeters

This section lists elements that serve to identify a particular installed component as the target of the containing step (for example, control service call). Note that not all targeters can be used with all targeted steps. Each targeter specifies the steps that it may be used with. The following table summarizes the targeters that may be used for the various targeted steps.

targeter	uninstall	call	check-Dependency	create-Dependency	add-Snapshot
installedComponent	yes	yes	yes	yes	yes
systemService	yes	yes	yes	yes	yes
systemType	yes	yes	yes	yes	yes
thisComponent 1	yes	yes	no	no	yes
superComponent 1	yes	yes	no	no	yes
nestedRef 2	yes	yes	yes	no	yes
allNestedRefs 2	yes	yes	no	no	yes
toplevelRef 2	yes	yes	yes	yes	yes
dependee 1	yes	yes	no	no	yes
allDependants 1	yes	yes	no	no	yes

1 may only be used for steps appearing within components

2 may only be used for steps appearing within composite components

The systemService Installed Component Targeter

The <systemService> element may be used as a targeter for the <checkDependency>, <createDependency>, <call>, <uninstall>, and <addSnapshot> steps. It identifies a particular system service component that is assumed to be installed on the current physical host.

Note – Use of the <systemService> targeter implicitly retargets to the root physical host of the current host. If one needs to target a system service on a different host, a <retarget> step must be used. One cannot otherwise name a new host within the <systemService> targeter.

installedComponent Targeter attributes

Name	Type	Required	Configurable	Description
name	entityName	Yes	No	The name of the installed component.
path	pathReference	No	No	The path of the component. If unspecified, the path of the containing entity is assumed.
version	version	No	No	The version of the installed component. If unspecified, the most recently installed component with any version is used.
versionOp	One of: "="	No	No	Specifies the operator to use when comparing the version attribute against versions of components installed on the target host. If more than one installed component applies, the most recently installed is used. If unspecified, ">=" is used by default. Ignored if version unspecified.
	">="			
	">"			

Name	Type	Required	Configurable	Description
only Compat	Boolean	No	No	If true, specifies that only components that are call compatible with the component having the named version should be matched. If true, a component with the named version must exist. If unspecified, "false" is used by default. Ignored if version unspecified.
installPath	String	No	Yes	The install path of the installed component. If unspecified, the most recently installed component in any path is used. The value is converted to universal format prior to component resolution.
host	string	No	Yes	The host on which the component is installed. Defaults to the current host. See the <retarget> step host attribute for details.

systemService Targeter attributes

Name	Type	Required	Configurable	Description
name	systemName	Yes	No	The name of the system service component.

The systemType Installed Component Targeter

The <systemType> element may be used as a targeter for the <checkDependency>, <createDependency>, <call>, <uninstall>, and <addSnapshot> steps. It identifies a component that is an instance of a particular type that is assumed to be installed on the target host. If more than one installed component matches the specified criteria, the one that was most recently installed is used.

The installedComponent Installed Component Targeter

The <installedComponent> element may be used as a targeter for the <checkDependency>, <createDependency>, <call>, <uninstall>, and <addSnapshot> steps. It identifies a particular installed component instance that is assumed to be installed on the target host.

Note – This targeter matches the named component directly, and cannot be used to match derived instances of the named component. Use the <systemType> targeter to target components derived from a particular type.

systemType Targeter attributes

Name	Type	Required	Configurable	Description
name	systemName	Yes	No	The name of the system type component.
installPath	String	No	Yes	The install path of the desired component. The value is converted to universal format prior to component resolution.
host	string	No	Yes	The host on which the component is installed. Defaults to the current host. See the <retarget> step host attribute for details.

The thisComponent Installed Component Targeter

The <thisComponent> element may be used as a targeter for the <call>, <uninstall> and <addSnapshot> steps. It specifies that the component containing the step should be used as target of the step. Only steps contained within a component may use this targeter. This element has no attributes.

Note – If the steps listed do not contain a component targeter element, then `<thisComponent>` is assumed by default.

The superComponent Installed Component Targeter

The `<superComponent>` element may be used as a targeter for the `<call>`, `<uninstall>` and `<addSnapshot>` steps. It specifies that the base component of the component containing the step should be used as target of the step. Only steps contained within a derived component may use this targeter. This element has no attributes.

Note – This targeter will always bind to the base component's definition of the step in question, even if the derived component overrides it.

The nestedRef Installed Component Targeter

The `<nestedRef>` element may be used as a targeter for the `<checkDependency>`, `<call>`, `<uninstall>` and `<addSnapshot>` steps. It identifies a nested component reference declared or inherited by the current composite component. Only steps within composite components may use this targeter.

It is assumed that the named component reference has already been installed by the calling component, and an error is raised otherwise. Note that if the nested component reference was installed on a host other than the current target host, use of the `<nestedRef>` targeter will implicitly retarget the associated step to that host.

nestedRef Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a nested component reference within this component.

The allNestedRefs Installed Component Targeter

The <allNestedRefs> element may be used as a targeter for the <call>, <uninstall> and <addSnapshot> steps. It identifies the set of all nested components references declared or inherited by the current composite component. Only steps within composite components may use this targeter.

Note – This targeter is special in that it may identify any number of components. If it identifies no components, then the step is a no-op. If it identifies more than one component, then the step is semantically expanded as if there is a separate occurrence of the step that using the <nestedRef> targeter for each component identified. The steps are executed serially rather than in parallel, and ordering of the steps varies depending on the step type, as described later. If execution of the step on one of the components causes an error, then the step will not be executed on the remaining matching components.

When used as a targeter for a <call> or <addSnapshot> step, the targeter matches all nested component references that are currently installed by this component, in the order of installation.

When used as a targeter for an <uninstall> step, the targeter matches all nested component references that are currently installed by this component, in the reverse order of installation.

The toplevelRef Installed Component Targeter

The <toplevelRef> element may be used as a targeter for the <checkDependency>, <createDependency>, <call>, <uninstall> and <addSnapshot> steps. It identifies a toplevel component reference declared or inherited by the current composite component. Only steps within composite components may use this targeter.

This targeter is semantically equivalent to the <installedComponent> targeter, except that the "name", "path" and "version" attributes are predefined based on the referenced component.

toplevelRef Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a toplevel component reference within this component.
versionOp	One of: ■ "=" ■ ">=" ■ ">"	No	No	Specifies the operator to use when comparing the version of the referenced component against versions of components installed on the target host. If more than one installed component applies, the most recently installed is used. If unspecified, ">=" is used by default.
only Compat	Boolean	No	No	If true, specifies that only components that are call compatible with the referenced should be matched. If unspecified, "false" is used by default.
installPath	String	Yes	Yes	The install path of the referenced component. If unspecified, the most recent installation of the referenced component in any path is used. The value is converted to universal format prior to component resolution.

Name	Type	Required	Configurable	Description
host	String	Yes	Yes	The host on which the referenced component is installed. Defaults to the current host. See the <retarget> step host attribute for details.



The dependee Installed Component Targeter

The <dependee> element may be used as a targeter for the <call>, <uninstall> and <addSnapshot> steps. It identifies an installed component that the calling component has a declared dependency on (created via <createDependency>). Only steps within components may use this targeter.

dependee Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a dependency created by this component.



The allDependants Installed Component Targeter

The <allDependants> element may be used as a targeter for the <call>, <uninstall> and <addSnapshot> steps. It identifies the set of installed components that have a declared dependency on the calling component (created through <createDependency>). Only steps within components may use this targeter.

This targeter functions similarly to the addNestedRefs targeter in that it causes the containing step to be mapped over all matching components. Order of the mapping over the dependant components is unspecified.

allDependants Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a dependency created on this component by other components.

Repository Component Targeters

This section lists elements that serve to identify a particular component that resides in the master server repository as the target of the containing step (e.g. install). Note that not all targeters can be used with all targeted steps. Each targeter specifies the steps that it may be used with. The following table summarizes the repository targeters that may be used for the various targeted steps.

targeter	install
component 1	yes
thisComponent 2	yes
superComponent 2	yes
nestedRef 3	yes
allNestedRefs 3	yes
toplevelRef 3	yes

1 may only be used for steps appearing within simple plan
2 may only be used for steps appearing within components
3 may only be used for steps appearing within composite components

The component Repository Component Targeter

The <component> element may be used as a targeter for the <install> step. It identifies a particular component that is assumed to exist in the CR repository. Only steps contained within a simple plan may use this targeter.

component Targeter Attributes

Name	Type	Required	Configurable	Description
name	entityName	Yes	No	The name of the component.
path	pathReference	No	No	The path of the component. If unspecified, the path of the containing entity is assumed.
version	version	No	No	The version of the component. If unspecified, the latest component is used.
host	string	No	Yes	The host on which the component should be installed. Defaults to the current host. See the <retarget> step host attribute for details.

The thisComponent Repository Component Targeter

The <thisComponent> element may be used as a targeter for the <install> step. It specifies that the component containing the step should be used as target of the step. Only steps contained within a component may use this targeter. This element has no attributes.

Note – If the steps listed do not contain a component targeter element, then <thisComponent> is assumed by default.

The superComponent Repository Component Targeter

The <superComponent> element may be used as a targeter for the <install> step. It specifies that the base component of the component containing the step should be used as target of the step. Only steps contained within a derived component may use this targeter. This element has no attributes.

Note – This targeter will always bind to the base component's definition of the step in question, even if the derived component overrides it.

The nestedRef Repository Component Targeter

The <nestedRef> element may be used as a targeter for the <install> step. It identifies a nested component reference declared or inherited by the current composite component. Only steps within composite components may use this targeter.

It is assumed that the named component reference has not been previously installed by the calling component, and an error is raised otherwise. Note that if a nested referenced component is to be installed on different host, a <retarget> step must be used. One cannot otherwise name a new host within the <nestedRef> targeter. It is not possible to install a nested component reference on more than one host for a given containing component.

nestedRef Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a nested component reference within this component.

The allNestedRefs Repository Component Targeter

The <allNestedRefs> element may be used as a targeter for the <install> step. It identifies the set of all nested components references declared or inherited by the current composite component. Only steps within composite components may use this targeter.

Note – This targeter is special in that it may identify any number of components. If it identifies no components, then the step is a no-op. If it identifies more than one component, then the step is semantically expanded as if there is a separate occurrence of the step that using the <nestedRef> targeter for each component identified. The steps are executed serially rather than in parallel, and ordering of the steps varies depending on the step type, as described later. If execution of the step on one of the components causes an error, then the step will not be executed on the remaining matching components.

When used as a targeter for an <install> step, the targeter matches all nested component references declared in this component, in the order of declaration.

The toplevelRef Repository Component Targeter

The <toplevelRef> element may be used as a targeter for the <install> step. It identifies a toplevel component reference declared or inherited by the current composite component. Only steps within composite components may use this targeter.

This targeter is semantically equivalent to the <component> targeter, except that the "name", "path" and "version" attributes are predefined based on the referenced component. A toplevel component reference may be installed any number of times on any number of hosts.

toplevelRef Targeter Attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of a toplevel component reference within this component.
host	String	No	Yes	The host on which the referenced component is to be installed. Defaults to the current host. See the <retarget> step host attribute for details.

Component Install-only Steps

This section lists steps that may only be used within an install block of a component (may be either simple or composite).

createDependency Step

A step used to create a persistent dependency of the current component on another component. When executed, this step first checks for an installed component matching the given criteria, and fails if no such component exists (just as with the <checkDependency> step). If match is found, then a persistent dependency is created between the matching component (the "dependee") and the calling component (the "dependant"). Note that if multiple installed components match the criteria (possible when no install path is specified), then the latest is used as the dependee. The persistent component is created with the name specified in the step. The name must be unique amongst all dependencies created in the install block.

Once created, the persistent dependency remains until the dependant component is uninstalled. Note that if the installation of the dependant component fails in a subsequent step after having created a persistent dependency, then the dependency is removed immediately at the time of failure.

A given component may depend on any number of other components by executing a <createDependency> step for each. It is recommended (but not required) that the createDependency steps appear as the first steps within an install block so that the installation fails prior to doing any real work if the dependencies are not satisfied.

createDependency Step attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of the dependency to create. Must be unique among all dependencies created by the current component.

createDependency Step Child elements

Name	How Many?	Description
installed component targeter	1	Identifies the dependee component.

createDependency Step Uninstall Implications

A component may not be uninstalled if there exist installed components that depend on it. When an uninstall block of a component is encountered, if there exists one or more persistent dependencies for which the component is the dependee, then the uninstall fails immediately. However, if a component B is being uninstalled by another component A, then dependencies created by A on B will not prevent B from being uninstalled, and will be implicitly removed when B is successfully uninstalled.

The dependee component can specify actions to uninstall its dependants using a <dependantCleanup> block.

createDependency Step Reinstall Implications

A component install is considered a reinstall if there is a pre-existing component in the same version tree that is installed on the same host and install path. A component may be reinstalled if and only if the new component also satisfies all the dependants of the original component. A component can always be reinstalled with the same version component, but can only be reinstalled with a different version if the new version also matches the constraints specified in the <createDependency> step that created the persistent component.

When a simple install block of a component is encountered, we determine if the install will overwrite an existing install. If so, then we find all persistent dependencies for which the existing component is the dependee, and re-verify that the constraints of the dependency are still satisfied with the new component being installed. If any are not satisfied, the install of the new component fails, and the original component remains installed. Otherwise, if and when the new component has successfully completed its install, it becomes the new dependee on all persistent dependencies. Also note that when the install of the new component successfully completes, then the original component is considered uninstalled, and all the persistent dependencies for which it was the dependant are removed. This implies that the new component is responsible for recreating dependencies as needed.

createDependency Step Naming Conventions

By convention, dependencies are named using the convention "xxx2yyy", where xxx in some way indicates the name of the dependant component, and yyy in some way indicates the name of the dependee component. For example, a WebLogic managed server might have a "server2domain" dependency on its admin server, and a "server2cluster" dependency on its containing cluster. This convention facilitates self-documentation of the nature of the dependency relationship, and makes it readable for both the dependee and dependant relationships of a particular component.

createSnapshot Step

A step used to create a snapshot of the current installed state of the component being installed. Any number of <createSnapshot> steps may appear within an install block.

Note – The named snapshot block may not declare any required parameters, as the createSnapshot step does not support argument passing. Argument passing is not supported because doing so would also require support for argument collection and passing at the time a comparison is later performed on the resulting snapshot.

When a comparison is performed on a composite component, it includes all snapshots created directly by that component, as well as the complete tree of snapshots created by the recursive installation of all nested component references. However, snapshots associated with toplevel component references are not considered, and therefore must be explicitly included in the snapshot of the composite component using the <addSnapshot> capture directive, as appropriate.

Also note that nested components may have some inter-dependencies that make it necessary to defer snapshot capture until all such components are installed. A nested component that deploys a directory and a nested component that deploys a file into that directory is an example of such an inter-dependence. In these cases, it is recommended that the containing component install the nested components using parameter passing or special install blocks that tell the nested component not to take the snapshot during their install, and then have the snapshot block of the containing component include appropriate snapshots of the nested components using the <addSnapshot> directive.

createSnapshot Step attributes

Name	Type	Required	Configurable	Description
blockName	entityName	Yes	No	The name of the snapshot block to execute within this component.

install Step

A step used to install a component onto a target host. It causes the steps of the named install block of the targeted component to be executed.

The syntax of this step is as specified for the simple plan <install> step, except that the component targeter may be omitted, in which case <thisComponent> is assumed.

When used within a component, this step is often used to call into another existing install block in the same component. In this case, the component is not considered installed until the outermost install block has completed its execution. Furthermore, the component is only considered installed on the host on which the install step was initially targeted, even if calls to other local install blocks were retargeted to other hosts.

This step can also be used within composite components to install referenced components. The referenced components may be installed on hosts other than that on which the containing component is being installed. Note that if the install of the containing component fails, then any nested referenced components that were successfully installed prior to the failure are implicitly uninstalled without executing an uninstall block. However, any toplevel referenced components that were successfully installed prior to the failure remain installed.

Simple Component Install-only Steps

This section lists steps that may only be used within an install block of a simple component.

deployResource Step

A step used to deploy the resource of the containing component. This step has no attributes or child elements.

Directory type resources whose `deployMode` is `ADD_TO` have each of their contained files copied, preserving directory structure and creating new directories as needed. Existing directory structure and content will not be changed (other than copying of resource content). Individual file permissions and ownership are updated as appropriate.

Directory type resources whose `deployMode` is `REPLACE` are treated the same as `deployMode ADD_TO`, except that any preexisting directory is recursively removed prior to deployment.

All other resources are copied, then have their permissions and ownership updated as appropriate. Resources checked in as configurable, will undergo variable substitution prior to being copied. Configurable resources may reference any variable that is accessible to the component in which the resource was declared.

Component Uninstall-only Steps

This section lists steps that may only be used within an uninstall block of a component.

uninstall Step

A step used to uninstall a component from a target host. It causes the steps of the named uninstall block of the targeted component to be executed.

The syntax of this step is as specified for the simple plan `<uninstall>` step, except that the component targeter may be omitted, in which case `<thisComponent>` is assumed.

When used within a component, this step is often used to call into another existing uninstall block in the same component. In this case, the component is not considered uninstalled until the outermost uninstall block has completed its execution. Furthermore, the component is only considered uninstalled on the host on which the uninstall step was initially targeted, even if calls to other local uninstall blocks were retargeted to other hosts.

This step can also be used within composite components to uninstall referenced components. Note that when a composite component is uninstalled, all of its nested referenced components are that were not explicitly uninstalled are implicitly uninstalled by the system without running an uninstall block. However, toplevel referenced components that were not explicitly uninstalled remain installed.

Simple Component Uninstall-only Steps

This section lists steps that may only be used within an uninstall block of a simple component.

undeployResource Step

This step has no attributes or child elements. A step used to remove the resource of the containing component.

Directory type resources whose deployMode is ADD_TO will have their files removed, but directories remain. Directory type resources whose deployMode is REPLACE will have the entire directory removed.

All other resources are treated as simple files and removed.

Note – The resource is removed regardless of whether or not it was originally deployed during the install of this component. In particular, even if this component was installed using an install block that did not contain a <deployResource> step, the <undeployResource> step would still cause the resource to be removed as if it had been originally installed by the component.

The N1 Service Provisioning System Software Plan Schema

This chapter describes the XML schema for N1 Service Provisioning System software plans.

For an overview of the provisioning software's XML schema architecture, see Chapter 1.

The executionPlan Element

The entire plan is enclosed by the <executionPlan> element.

In the following tables, the “Config” column indicates whether or not a specific attribute may include substitution variable references in the form “[varName]”.

executionPlan Element attributes

Name	Type	Required	Description
xmlns	String	Yes	Required value: http://www.sun.com/schema/SPS
xmlns:xsi	String	Yes	Required value: http://www.w3.org/2001/XMLSchema-instance

xsi:schemaLocation	String	No	Recommended value: http://www.sun.com/schema/SPS plan.xsd
name	entityName	Yes	The name of the plan
path	pathName	No	The absolute path of the execution plan. If unspecified, the root path "/" is used by default.
description	String	No	The description of the plan
version	schemaVersion	Yes	The version of the plan schema being used. The only value currently permitted is 4.0.

executionPlan child elements

Plans are either simple or composite. Simple plans do not contain or call other plans. A simple plan is a sequential list of steps that will be executed on a particular set of target machines. Composite plans are composed solely of other subplans. Composite plans are not directly targeted, as each simple subplan they recursively contain has a different set of targets.

Name	How Many?	Description
paramList	0 or 1	A list of parameters for use within the plan.
varList	0 or 1	A list of variables for use within the plan.
simpleSteps	0 or 1	Contains a list of simple steps. Either this element or the compositeSteps element must be present, but not both.
compositeSteps	0 or 1	Contains a list of composite steps. Either this element or the simpleSteps element must be present, but not both.

The Plan paramList Element

The plan <paramList> element is a child of the <executionPlan> element and is used to declare a list of parameters for use by the steps contained in the plan and any components they reference. When this plan is run as a top-level plan, the caller will be prompted to enter values for all parameters declared in this list. When this plan is invoked as the result of an <execSubplan> step in another plan, the calling plan must explicitly pass values for all non-default parameters declared in the <paramList>.

Plan paramList Element child elements

Name	How Many?	Description
param	1 or more	A plan parameter declaration, including name, prompt, and default value.

The Plan param Child Element

The plan <param> element is a child of the plan <paramList> element and is used to declare a parameter for use within the plan.

Plan param Child Element attributes

Name	Type	Required	Description
name	identifier	Yes	The name of the plan parameter. The name must be unique amongst all the top-level plan parameters and variables.
prompt	String	No	The UI text to display when prompting for the value of the parameter. If unspecified, name is used by default.
default	String	No	The default value of the parameter. The default value may not include references to other parameters or variables.

display Mode	One of: PASSWORD CLEAR BOOLEAN	No	The display mode of the variable. If unspecified, CLEAR is used by default. If PASSWORD, the client-entered value will be hidden (that is, not shown or replaced with ***). If BOOLEAN, the parameter is entered as a checkbox. Otherwise, if CLEAR or BOOLEAN, it is safe for the value to be displayed as entered.
--------------	---	----	---

The Plan varList Element

The plan <varList> element is a child of the <executionPlan> and <inlineSubplan> elements and is used to declare a list of variables for use by the steps contained in the plan and any components they reference. The values of the variables are defined at the point of declaration, and cannot be redefined.

Plan varList Element

Name	How Many?	Description
var	1 or more	A plan variable declaration, including name and value.

The Plan var Child Element

The plan <var> element is a child of the plan <varList> element and is used to declare a plan variable including name and value.

Plan var Child Element attributes

Name	Type	Required	Configurable	Description
name	identifier	Yes	No	The name of the local variable. The name must be unique amongst every variable in the containing <varList>. Variables associated with the top-level <executionPlan> must also be unique amongst the plan parameters.
default	String	Yes	Yes	The default value of the plan variable, which may include references to other plan variables declared earlier, and plan parameters. Additionally, if this plan is a simple plan, then references target host attributes and installed component variables may also be included.

The simpleSteps Element

The <simpleSteps> element is a child of the <executionPlan> and <inlineSubplan> elements and contains one or more “shared” or “simple plan only” steps. The presence of a <simpleStep> element indicates that the plan is a simple (as opposed to composite) plan. When run, the steps within this element will be sequentially executed on a set of logical target hosts chosen by the caller.

simpleSteps Element attributes

Name	Type	Required	Description
------	------	----------	-------------

executionMode	One of: PARALLEL SERIES	No	Whether the contained steps should be executed in series or in parallel over the target hosts. If unspecified, PARALLEL is used.
limitToHostSet	String	No	The name of the host set containing the hosts that are considered valid logical targets for this plan, as described below.

The simpleSteps limitToHostSet Attribute

The `limitToHostSet` attribute of the `<simpleSteps>` element specifies the name of the host set containing the hosts that are considered valid targets for this plan. If unspecified, all hosts are considered valid targets. Otherwise, the targets specified by the client must be a subset of the hosts contained in the named host set. If the targets include a host not contained in the named host set, it is a plan run time error. It is also a plan run time error to specify a name that does not correspond to an existing host set. The plan run time errors described here are reported during validation before preflight starts.

Child Elements of the simpleSteps Elements

The `<simpleSteps>` element children consist of one or more “shared” or “simple plan only” steps. These steps may include references to plan parameters and variables.

See “Simple Plan-only Steps” on page 141 for details.

The compositeSteps Element

The `<compositeSteps>` element is a child of the `<executionPlan>` and `<inlineSubplan>` elements and contains one or more “composite plan-only” steps. The presence of a `<compositeStep>` element indicates that the plan is a composite (as opposed to simple) plan. The `<compositeSteps>` element does not contain any attributes.

Child Elements of the compositeSteps Element

The `<compositeSteps>` element children consist of one or more “composite plan-only” steps. These steps may include references to plan parameters and variables.

See “Simple Plan-only Steps” on page 141 for details.

Child Elements of the compositeSteps Element

The <compositeSteps> element children consist of one or more “composite plan- only” steps. These steps may include references to plan parameters and variables.

See “Simple Plan-only Steps” on page 141

Shared Steps

This section lists steps that may be used in either a component or a simple plan. These steps include:

- <call> (described in “call Step” on page 111)
- <checkDependency> (described in “CheckDependency Step” on page 113)
- <execJava> (described in “execJava Step” on page 113)
- <execNative> (described in “execNative Step” on page 115)
- <if> (described in “if Step” on page 124)
- <pause> (described in “pause Step” on page 125)
- <processTest> (described in “processTest Step” on page 125)
- <raise> (described in “raise Step” on page 126)
- <reboot> (described in “reboot Step” on page 127)
- <retarget> (described in “retarget Step” on page 127)
- <sendCustomEvent> (described “The sendCustomEvent Step” on page 130)
- <transform> (described in “transform Step” on page 130)
- <try> (described in “try Step” on page 134)
- <urlTest> (described in “urlTest Step” on page 137)

In the following tables, the “Config” column indicates whether or not a specific attribute may include substitution variable references in the form “:[varName]”.

call Step

You can use the <call> step to execute a control block associated with a component installed on the target host.

call Step attributes

Name	Type	Required	Configurable	Description
------	------	----------	--------------	-------------

blockName	String	Yes	No	The name of the control block to execute on the installed component.
-----------	--------	-----	----	--

call Step child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the control block.
installed component targeter	0 or 1	Identifies the component containing the control block to execute. If unspecified, <thisComponent> is used.

The argList Child Element

The <argList> element is a child of the <call>, <install>, <uninstall>, <execSubplan>, and <addSnapshot> steps. It specifies a list of variables to be passed as arguments to the called service. The called service declares the variables that it expects using a <paramList> element. The variables within the <argList> and the called <paramList> need not be the same. However, for any variable in the called <paramList> that does not have a default value, there must be a variable with the same name in the <argList>, or a preflight error is raised at plan runtime. For each variable in the called <paramList> for which there is a corresponding variable in the <argList>, the value of the variable in <paramList> will obtain the value of corresponding variable in <argList> for the duration of the execution of the called service.

Variables in <argList> that do not correspond to a variable in the called <paramList> will be silently ignored. This permits services to be redefined with additional parameters, while still allowing for backward compatibility such that one plan can call both old and new versions of the service.

The arguments of the <argList> element are expressed as attributes. For example, the following <argList> declares two arguments, "password" and "path":

```
<argList password=":[password]"
path="/tmp"/>
```

Order is not important.

argList Child Element attributes

The <argList> element must have at least one attribute. Each attribute is treated as a named variable to be passed to the called service. The name of each attribute must be an identifier without substitution variable references, and should correspond to the name of a parameter in the called service. The value of each attribute is an arbitrary string which may include references to variables in the enclosing scope, but not other arguments within the <argList>.

CheckDependency Step

You can use <checkDependency> to verify that a particular component has been installed on the target host. If an appropriate component has not been installed, the step fails and execution stops.

The dependency is checked using the contained component targeter. If the targeter successfully resolves a component, the dependency is considered satisfied. Otherwise, it is considered failed.

checkDependency Step child elements

Name	How Many?	Description
installed component targeter	1	Identifies the component to check for dependency

execJava Step

A step used to execute a Java™ Executor instance on the target host. If the executor instance raises an exception, the step fails and execution stops.

Note – The <execJava> step is an advanced feature intended only for internal use.

execJava Step attributes

Name	Type	Required	Configurable	Description
className	String	Yes	Yes	The full name of a public class with a public no-arg constructor that implements the ExecutorFactory interface (as specified by the Plan Executor).
classPath	String	No	Yes	The classpath containing the class named by the className attribute. If unspecified, the system class path of the remote agent is used. The classpath format is a semi-colon separated list of absolute paths to jar files on the agent.
timeout	positiveInteger	No	No	Specifies the number of seconds to wait for the command to complete before timing it out. If not specified the plan's exec native timeout period applies. Should be a value greater than 0.

Note – When an execJava step is contained within a component, it typically calls classes which are contained within one or more resources deployed by that component. When contained within a plan, the classes called are already resident on the agent, either as a system class of the agent itself, or more probably as a resource that was deployed with an existing component.

execJava Child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the Executor instance.

execNative Step

The <execNative> step is used to execute a native command on the target host. If the command produces an unexpected result, the step fails, and execution stops.

execNative Step attributes

Name	Type	Required	Configurable	Description
userToRunAs	String	No	Yes	The user to run this command as. If specified, the command is executed as this user; otherwise it's run as the defaultUserToRunAs in the config file. The value can be either a string username or a numeric UID. It should be a non-empty string.
dir	String	No	Yes	The absolute path to the working directory for the command. If not specified defaults to agent-specific configurable directory. Should be a non-empty string
timeout	positive Integer	No	No	Specifies the number of seconds to wait for the command to complete before timing it out. If not specified the plan's exec native timeout period applies. Should be a value greater than 0.

execNative can have nested elements to specify the:

- Environment variables
- Background execution
- Command standard output file
- Command error output file
- Command input
- Command to execute
- Step success criteria

execNative Element child elements

Name	How Many?	Description
env	0 or more	Environment variables for the child process
background	0 or 1	Specifies that the command should be executed as a background process
outputFile	0 or 1 1	The name of the file in which to store the command's standard output
errorFile	0 or 1 1	The name of the file in which to store the command's standard error output.
inputText	Only 1 of these elements may be specified 2	The text to feed to the command's standard input. This element cannot be specified if inputFile is specified.
inputFile		The name of the file from which to read command's standard input
exec	1 of these elements must be specified 3	Specifies the executable to execute
shell		Specifies the shell command to execute
successCriteria	0 or 1	The criteria to evaluate if this step was successful or not.

1 Both <outputFile> and <errorFile> must be specified if <background> element is specified.

2 inputText and inputFile elements are optional. Only one of them may be specified at a time.

3 One of exec or shell elements must be specified. It is illegal to have both exec and shell elements in an execNative element.

The <inputText> and <inputFile> elements are optional. Only one of them may be specified at a time.

One of `exec` or `shell` elements must be specified. It is illegal to have both `exec` and `shell` elements in an `execNative` element.

The `env` Child Element

The `<execNative>` element may optionally have `<env>` elements to specify environment variables for the command. You can use `<env>` to supply new variables for the command's environment or to over-ride existing variables.

The set of the command's environment variables will be a union of the set of Remote Agent's environment variables and the variables supplied using the `<env>` tags.

`env` Element attributes

Name	Type	Required	Configurable	Description
name	String	Yes	Yes	The name of environment variable. Should be a non-empty string
value	String	Yes	Yes	The value of the environment variable. The value may contain <code>\${var_name}</code> strings to refer to the values of RA's environment variables. If <code>var_name</code> refers to a variable over-ridden using the <code>env</code> tag, its substituted value the one defined in RA's environment, not the over-ridden one. If <code>\$</code> needs to appear literally <code>\${}</code> may be used. All instances of <code>\${}</code> will be replaced by <code>\$</code> .

background element

The `<background>` element is a child of the `<execNative>` element and, when present, specifies that the command should be executed as a background process. It has no attributes or child elements.

For an `<execNative>` with a `<background>` element, the following constraints apply.

- `<successCriteria>` need not be specified. The step will succeed if there were no issues starting off the background process. If specified, the `<successCriteria>` is tested against the script that is used to run the command as a background process.
- No standard output or error output will be captured for the executed command. Clicking on details will show exist status as 0 and the empty output and error output. However in case there were problems starting the native command in background, you will see a different exit status, depending on the nature of the

error and diagnostic output.

- The <outputFile> and <errorFile> elements must be specified. If the specified files already exist, they will be overwritten.

outputFile element

The outputFile element is used to specify the path to a local file on the agent in which the standard output of the command being executed should be stored. The path to the local file is specified through the name attribute in the outputFile element. Relative paths are interpreted relative to the command's working directory.

The outputFile element must be specified if the background element is specified.

If outputFile is not specified and the successCriteria element does not specify an outputMatches, the respective output of the command will not be stored and will be lost. If outputMatches is specified, the standard output will be stored in a temporary file and that will be deleted after command is done executing.

outputFile Attributes

Name	Type	Required	Configurable	Description
name	String	Yes	Yes	The file to write the standard output of the command to. Should be a non-empty string.

errorFile element

The errorFile element is used to specify the path to a local file on the agent in which the error output of the command being executed should be stored. The path to the local file is specified through the name attribute in the errorFile element. Relative paths are interpreted relative to the command's working directory.

The errorFile element must be specified if the background element is specified.

If errorFile is not specified and the successCriteria element doesn't specify an errorMatches, the respective output of the command will not be stored and will be lost. If errorMatches is specified, the error output will be stored in a temporary file and that will be deleted after command is done executing.

errorFile Attributes

Name	Type	Required	Configurable	Description
name	String	Yes	Yes	The file to write the error output of the command to. Should be a non-empty string.

The inputText Child Element

The `<inputText>` element is a child element of `<execNative>`. The `<inputText>` element specifies any arbitrary text that should be fed into the command's standard input. The text is specified as contents of this element.

You can use `<inputText>` to enclose input in CDATA sections, preserving the formatting of the input and avoiding parse errors that might be caused by input containing the characters `'&` and `'<`.

For example:

```
<execNative>
  <inputText>                ls -l |fgrep '*test*' |sort -u >file.out
</inputText>                <command exec="sh"/> </execNative>
```

If the `<inputText>` is specified, it will always be enclosed within a CDATA section when generating XML for the `<execNative>` step. All the characters within `inputText` are fed as it is to the command being executed. If `inputText` contains only white spaces, those white spaces will be fed, exactly as they are specified, into the standard input of the command being executed.

The contents of `<inputText>` are config-generated.

The `<inputText>` element has no attributes.

inputFile Child Element

The `<inputFile>` element is a child element of `<execNative>`. The `<inputFile>` element specifies the path to a local file on the Remote Agent whose contents should be fed into the standard input of the command being executed. The path to the local file is specified through the `name` attribute in the `<inputFile>` element.

Note – The `<inputFile>` element cannot be used with the `<inputText>` element in an `<execNative>` command.

inputFile Element attributes

Name	Type	Required	Configurable	Description
name	String	Yes	Yes	The file to read the input of the command from. Should be a non-empty string. If relative path is specified, it is taken relative to the command's working directory.

The exec Child Element

An `<execNative>` step can contain exactly one `<exec>` element. The `<exec>` element specifies the details of the native command to be executed.

The `<exec>` element contains:

- an attribute, `cmd`, that specifies the name of the command to execute
- a set of nested `arg` elements that specify each of the arguments for the command specified with `cmd`.

`<execNative>` executes the command specified in `cmd` directly with the arguments in the order they are specified.

For example, the `<execNative>` step below:

```
<execNative>
  <exec cmd="ps">
    <arg value="-fu"/>
    <arg value="sps"/>
  </exec>
</execNative>
```

will execute the command: `ps -fu sps`.

exec Element attributes

Name	Type	Required	Configurable	Description
------	------	----------	--------------	-------------

cmd	String	Yes	Yes	The path of the command to execute. If the specified path is not absolute the command is looked up using the platform specified PATH environment variable setting for the RA. Should be a non-empty string
-----	--------	-----	-----	--

exec Element child elements

Name	How Many?	Description
arg	0 or more	The arguments to the command being executed. Each arg element defines one positional parameter to the command.

arg Child Element attributes

Name	Type	Required	Configurable	Description
value	String	Yes	Yes	The argument value. This argument will be supplied as the nth argument to the command where arg is the nth child of the command element.

The shell Child Element

The <shell> element is a child element of <execNative>. The <shell> element specifies the command to be executed as its content. The command to be executed is interpreted using an interpreter specified using the cmd attribute. The command thus specified is executed using `sh -c 'command'` syntax for the platform. In this form the cmd attribute has to be specified to indicate the shell command to use to execute the command.

For example:

```
<execNative>
  <shell cmd="/usr/bin/bash -c">
    ls -l |fgrep '*test*' |sort -u >file.out
  </shell>
</execNative>
```

This code executes the command:

```
/usr/bin/bash -c 'ls -l |fgrep '*test*' |sort -u >file.out'
```

To preserve formatting and to avoid XML parsing problems, the text contents of the command will always be enclosed within a CDATA element when generating XML representation from the <execNative> step.

It is illegal for command string to be empty or have only white space characters. The command string is supplied exactly as it is specified (with surrounding white space) to the shell.

The contents of the <shell> element are config-generated.

Attributes of the shell Element

Name	Type	Required	Configurable	Description
cmd	String	Yes	Yes	The shell command in the 'sh -c' syntax. The string should not contain any embedded quote characters. The string will be parsed to retrieve the shell name and the arguments using white space as delimiters. For example '/usr/bin/bash -c' should be a non-empty string

The successCriteria Element

The <successCriteria> element is a child element of <execNative>. The <successCriteria> element specifies the criteria to be used to evaluate whether or not an <execNative> step executed successfully. If this element is not specified, the default value is:

```
<successCriteria status="0" />
```

The successCriteria is ignored if an empty successCriteria is specified. If you specify the tag:

```
<successCriteria/>
```

The step will always succeed no matter what output or exit code the command generated.

successCriteria Element attributes

Name	Type	Required	Configurable	Description
status	integer	No (If more than one is specified, then these conditions are AND-ed together.)	No	The desired exit status of the command. Should be a positive integer.
outputMatches	String		Yes	The regular expression to match with the standard output generated by the command. Should be a non-empty string
errorMatches	String		Yes	The regular expression to match the standard error generated by the command. Should be a non-empty string
inverse	Boolean	No	No	If set to true the conditional expressed by echo element of success criteria is negated. The default value is false. That is the step succeeds only if the conditions specified through each attribute of successCriteria are not met

If the inverse attribute is set to true, it negates each of the conditions specified within <successCriteria>.

For example, an <execNative> step with the following successCriteria

```
<successCriteria status="1" outputMatches="bin" errorMatches="none" inverse="true"/>
```

will succeed if status is not 1 and output does not match 'bin' and error does not match 'none'.

<successCriteria> element that contains only the inverse attribute will be saved without the inverse attribute. That is if you specify

```
<successCriteria inverse="true"/>
```

It will be stored as:

```
<successCriteria/>
```

The result will be that <successCriteria> is ignored.

if Step

A step used conditionally execute a block of steps. It has no attributes, and children consist of a <condition>, <then>, and <else> element.

If the content of the <condition> element evaluates to true, then the steps of the <then> block are executed, otherwise the steps of the <else> block are executed, if present.

if Step Child Elements

Name	How Many?	Description
condition	1	The condition to evaluate
then	1	The steps to execute if condition is true
else	0 or 1	The steps to execute if condition is not true

if Step Example

The following is an example of an <if> step used to conditionally restart.

```
<if>
  <condition><istrue value=":[restart]"/></condition>
  <then>
    <call blockName="restart"/>
  </then>
</if>
```

condition Element

A child element of the <if> step that specifies a boolean expression. It has no attributes, and must contain exactly one boolean operator child element. See the Boolean Operators section for details of the permitted boolean operators.

then Element

A child element of the <if> step that specifies the steps to execute if the associated condition is true. It may contain any number of steps that are permitted within the scope of the block containing the <if> step.

else Element

A child element of the <if> step that specifies the steps to execute if the associated condition is not true. It may contain any number of steps that are permitted within the scope of the block containing the <if> step.

pause Step

The <pause> step pauses the execution of a plan for a specified amount of time. You can use <pause> to make a plan wait for required services to come online after being started.

pause Step attributes

Name	Type	Required	Configurable	Description
delaySecs	positiveInteger	Yes	No	The number of seconds to wait.

processTest Step

The <processTest> step is used to verify that a particular process is running on the target host. If the desired process does not exist, the step fails and execution stops.

Note – This step is intended for use with UNIX systems.

processTest Step attributes

Name	Type	Required	Configurable	Description
delaySecs	positiveInteger	Yes	No	The number of seconds to wait before testing if the process exists.
timeoutSecs	positiveInteger	Yes	No	The number of seconds to wait for the process to come online before failing. This time starts after the delay has completed.
processNamePattern	String	Yes	Yes	A glob-style pattern to use to match the desired process name.
user	String	No	Yes	A glob-style pattern to use to match the name of the process owner. If unspecified, the process owner is not considered as part of the test.

raise Step

A step that always fails (though it may be caught and handled in a <try> step). It contains a single attribute "message", and has no children.

The <raise> step is used to indicate a failure condition without having to construct an artificial step to do so. It most often appears within a <catch> block to propagate an error condition after cleaning up.

raise Step attributes

Name	Type	Required	Configurable	Description
message	string	No	Yes	A message describing the error condition. Defaults to a generic, system specified message.

raise Step Example

The following example shows how the <raise> step is used to repropagate an error condition from within a <catch> block, after noting the error in a log.

```

<control blockName="default">
  <try>
    <block>
      <!-- some arbitrary processing here -->
    </block>
    <catch>
      <!-- note error in log -->
      <execNative>
        <exec cmd="appendLog"> <arg value="an error occurred"/> </exec>
      </execNative>
      <!-- rethrow error -->
      <raise/>
    </catch>
  </try>
</control>

```

reboot Step

A step that causes the agent to reboot before proceeding with the rest of the plan. It may only be used on windows based platforms. If encountered on a non-windows platform, an error is raised. It is also an error to reboot an agent that resides on the same host as the master server. The errors discussed here are preflight errors.

reboot Step attributes

Name	Type	Required	Configurable	Description
timeout	positive integer	No	No	The maximum number of seconds to wait for the machine to come back up. If not specified the plan's execNative timeout period applies.

retarget Step

A step that changes the execution target for a set of steps. Retarget steps may be nested.

retarget Step attributes

Name	Type	Required	Configurable	Description
host	String	Yes	Yes	The host on which the contained steps should be executed.

retarget Step Child Elements

Name	How Many?	Description
varList	0 or 1	Local variables available to the contained steps. These are evaluated within the scope of the new host.
steps	0 or more	The steps to execute on the new host. May be any step permitted within the scope of the block containing the <retarget> step.

retarget Step host Attribute

The "host" attribute is used in the <retarget> step, as well as in various component targeter elements. Its value is the name of a host, which may include substitution variable references. It may also include symbolic names "/" to reference the root physical host of the current execution target, or "../..*" to reference a parent host of the current execution target. The complete syntax supported is specified in the "host redirects" section of Chapter 16.

Note – When a host attribute is specified within a component targeter, it is semantically equivalent to enclosing the containing step within a retarget step.

retarget Step Execution Semantics

When a <retarget> step is encountered, we first evaluate the "host" attribute in the context of the current host of the caller.

If the value of the "host" attribute is different from the name of the current host, then the following steps are taken:

1. We resolve the host name to an actual host. If no such host exists, an error is raised.
2. We verify that the current user has "plan run" permission on the the given host. If not, an error is raised.
3. We verify that the root physical host of the host contains an RA, can be connected to, and is updated and prepared. If not, an error is raised.
4. We obtain a lock on the host, while retaining locks on all previously visited hosts. If the current execution thread already locks the host, this operation is effectively a no-op. If the host is already locked by another execution thread, this operation will block until the host is unlocked. If the request for a lock would result it a deadlock, then an error is raised.
5. The host becomes the new "current" host. The "physical" host is reset based on the new "current" host. The "initial" host does not change.

Once the above has completed, variables of the <varList> element are evaluated in the context of the new current host. Local variables may hide variables of enclosing scopes. Then each of the steps is executed in the context of the new current host.

Finally, if the retarget operation changed the current host, then it is unlocked as appropriate and the current host is reset to the current host of the caller. Note that if the host was locked previously in the current execution thread, then it remains locked until the block that first acquired the lock completes.

Note – An empty <retarget> block is still useful as a mechanism to prevalidate that the current user has appropriate permissions, and that the host is properly prepared.

retarget Step Example

The following is an example of a "restart" control service one might expect to find on a WebLogic managed server. It is implemented by calling a control on the admin server to "stop" the managed server, and then making a call on the local machine to start the server.

Note – The "adminHostName" variable is evaluated on the current host of the caller (which is assumed to be the vhost containing the managed server). The domainName variable is evaluated on the retargeted host (which is assumed to be the vhost containing the admin server). The ADMIN_SERVER component is also resolved on the retargeted host.

```
<control name="restart">
  <varList>
    <var name="adminHostName" default=":[target:adminHostName]"/>
  </varList>
  <retarget host=":[adminHostName] ">
```

```

        <varList>
            <var name="domainName" default=":[target:domainName]" />
        </varList>
        <call blockName="stopServer">
            <argList serverName=":[serverName]"
domainName=":[domainName]" />
            <installedComponent name="ADMIN_SERVER" />
        </call>
    </retarget>
    <call blockName="start" />
</control>

```

The sendCustomEvent Step

The <sendCustomEvent> step is used to generate a custom event with a particular message. This step can be used in conjunction with the notification rules module to send an email any time this step is encountered on a particular host.

sendCustomEvent Step attributes

Name	Type	Required	Configurable	Description
message	String	Yes	Yes	The message to include as the event text.

transform Step

The <transform> step is used to perform a text-based transformation on a file contained on the target host. Currently, Perl5 and XSLT based transforms are supported.

transform Step attributes

Name	Type	Required	Configurable	Description
input	String	No	Yes	The generalized path of the file on the target host to apply the transform to. If unspecified, input is read from the output file.
output	String	Yes	Yes	The generalized path of the file on the target host to write the result of the transform to.

The input and output attributes can reference the same or distinct files. The value of the input and output attributes is a generalized path that can include zip archives (or zip derivatives such as JAR, etc.) as directory elements, e.g. `webapp/myapp.jar/Configurable.xml`.

transform Element child elements

The `<transform>` element children specify the transformation to be applied to the input file. The child elements can be any one of the following:

- A single `<stylesheet>` element defining an XSLT transform to apply on the input source.
- One or more `<subst>` elements defining Perl5-based substitution patterns to apply sequentially on the input source.
- A single `<source>` element naming an external file that contains the transform.
- Empty, meaning the content of the input file is copied directly to the output file. This is useful for extracting from or writing to zip archives.

The stylesheet transform Child Element

The `<stylesheet>` element is a child of the `<transform>` step, and specifies an XSLT transform to apply on the input source. At most one `<stylesheet>` element may appear as a child of a particular `<transform>` element, and they may not be used in conjunction with any other child elements.

The `<stylesheet>` element is an XSLT version 1.0 element as defined by the namespace `"http://www.w3.org/1999/XSL/Transform"`. Please see the XSLT specification at `"http://www.w3.org/TR/xslt"` for details. Note that only the XSLT `<stylesheet>` element is accepted as a child of the transform element. In particular, neither the XSLT synonym `<transform>` nor the simplified XSLT transform syntax described in section 2.3 of the XSLT specification are supported as children of the transform element.

The stylesheet element body may include substitution variable references so long as the body is still considered valid XSLT without first undergoing variable substitution.

When a <stylesheet> element is used as a transform, it is assumed that the input file is written in XML.

Example of the stylesheet Element

The following is an example of an XSLT transform.

```
<transform output="/etc/hosts">
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<xsl:for-each select="a" >
<xsl:value-of select="b"/>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
</transform>
```

The subst transform Child Element

The <subst> element is a child of the <transform> step, and specifies a Perl5-based substitution pattern to apply as a transform. Multiple <subst> elements may appear as children of the <transform> elements, but they may not be used in conjunction with any other child elements. When more than one <subst> elements appear, they are applied sequentially.

All occurrences of the pattern in the input file are replaced, including multiple occurrences within a line.

The details of the supported syntax can be found in the Oro documentation (class `org.apache.oro.text.regex.Perl5Substitution`).

subst Element attributes

Name	Type	Required	Configurable	Description
match	String	Yes	Yes	A case-sensitive Perl5 regular expression which is sought for in the input.

replace	String	Yes	Yes	A Perl5 replacement value substituted for each occurrence of the pattern given by "match". This value is not interpreted verbatim: the construct \$n is interpreted as the nth parenthesized expression inside the matching expression.
---------	--------	-----	-----	---

An Example of the subst Element

The following transform converts all occurrences of the string 127.0.0.xxx into 10.10.0.xxx within the file /etc/hosts:

```
<transform output="/etc/hosts">
  <subst match="127\.\0\.\0\.\(\d+)" replace="10.10.0.$1" />
</transform>
```

The source transform Child Element

The <source> element is a child of the <transform> step, and specifies an external file on the target host which contains the transform to be applied to the input file. At most one <source> element may appear as a child of a particular <transform> element, and they may not be used in conjunction with any other child elements.

Note – No config generation is performed on the named source file as part of the <transform> step. However, the named source file may be a config-type resource file deployed as part of a component install, in which case substitution variables contained in the source file would have been substituted when the file was deployed.

source Element attributes

Name	Type	Required	Configurable	Description
type	One of: PERL XSLT	Yes	No	The type of transform contained in the named file, as described below.

name	String	Yes	Yes	The name of the file on the target host that contains the transform. The content of the named file must correspond to the type defined by the "type" attribute. The name may not include zip archives as directory elements.
------	--------	-----	-----	--

The type Attribute of the source Element

The "type" attribute of the <source> element specifies the type of transformation contained in the named file.

A value of "PERL" indicates a Perl5-based transform similar to that of the <subst> element. In this case, the named file should have format similar to the following:

```
<?xml version='1.0'?>
<transform>
  <subst match="127\.\0\.\0\.\(\d+)" replace="10.10.0.$1" />
</transform>
```

Perl-type external transform files may contain any number of <subst> elements.

A value of "XSLT" indicates an XSLT transform. In this case the named file contains a standard XSLT version 1.0 transform as defined by the namespace "http://www.w3.org/1999/XSL/Transform". Note that unlike inline transforms which only allow the XSLT <stylesheet> element, XSLT transforms contained in external source files may include any valid top-level XSLT transform element including <stylesheet>, <transform>, and simplified XSLT syntax as described in section 2.3 of the XSLT specification.

try Step

The <try> step is used to specify typical error handling and cleanup logic for a block of steps. It has no attributes, and children consist of a <block>, <catch>, and <finally> element.

The <try> step is executed as follows:

- The steps in the <block> element are executed in order until all have completed or a step fails (as per the plan executor's definition of failure for each step).
- If and only if a <catch> element is defined and the execution of the block element terminated with a step failure other than plan abort or plan timeout, then the steps in the <catch> element are executed in order until all have completed or a step fails.

- If a <finally> element is defined, its steps are executed in order until all have completed or a step fails. This happens regardless of the success of the previous two elements unless either of the two previous elements failed due to plan abort or plan timeout.
- A corollary of the above rules is that if any contained step fails due to plan abort or plan (not step) timeout, then no other contained steps are executed, and execution of the <try> step ends with a failure.

The <catch> element is used to suppress and recover from errors encountered in the <block> element. The <finally> element is used to unconditionally perform some cleanup, regardless of whether typical errors were encountered.

The <try> step either succeeds or fails as follows:

- If the <try> step contains just a <finally> element (but no <catch>), then it fails if the execution of either the <block> or the <finally> element fails.
- If the <try> step contains just a <catch> element (but no <finally>), then it fails if and only if
 - the <catch> element executes and fails, or
 - the <block> element fails due to plan abort or plan timeout
- If the <try> step contains both a <catch> and a <finally> element, then it fails if and only if
 - the <catch> or the <finally> element executes and fails,
 - the <finally> element executes and fails, or
 - the <block> element fails due to plan abort or plan timeout

The critical point of the above is that failures in the <block> element are suppressed by the presence of a <catch> element.

try Step Child Elements

Name	How Many?	Description
block	1	The steps executed initially.
catch	0 or 1	The steps to execute in case of a typical error. Must be specified if finally is not.
finally	0 or 1	The steps to execute regardless of typical errors. Must be specified if catch is not.

try Step Examples

Suppose you have a component whose installation consists of resource deployment followed by a restart. In this case, the component is really considered installed after its resources are deployed, and a failure during restart should not affect its installed state. Typical errors can be suppressed during the restart as follows:

```
<installSteps blockName="default">
  <deployResource/>
  <try>
    <block>
      <call blockName="restart"><thisComponent/></call>
    </block>
    <catch/><!-- suppress all typical errors -->
  </try>
</installSteps>
```

Try blocks can also be used to model intelligent auto-upgrades. Suppose you are working with version 1.1 of a component, and you have two different install routines depending on whether version 1.0 was previously installed or not (fresh install versus upgrade install). On can model this in a single install block, as follows:

```
<installSteps blockName="default">
  <try>
    <block>
      <checkDependency>
        <installedComponent name="foo" version="1.0"/>
      </checkDependency>
      <!-- 1.0 install exists, do upgrade -->
    </block>
    <catch>
      <!-- 1.0 install doesn't exist, do fresh install -->
    </catch>
  </try>
</installSteps>
```

The `<finally>` block is most often used to cleanup temporary resources. The following example creates a temp file, processes it and then removes it.

```
<control blockName="default">
  <varList>
    <var name="file" default="/tmp/file.txt"/>
  </varList>
  <execNative outputFile=":[file]">
    <exec cmd="ls"><arg value="-l"/></exec>
  </execNative>
  <try>
    <block>
      <!-- process file in some way -->
    </block>
    <finally>
      <execNative>
        <exec cmd="rm"><arg value=":[file]"/></exec>
      </execNative>
    </finally>
  </try>
</control>
```

```
        </finally>
    </try>
</control>
```

block Element

The `<block>` element is a child element of the `<try>` step that specifies the primary steps executed by the `<try>` step. It contains one or more steps that are permitted within the scope of the block containing the `<try>` step.

catch Element

The `<catch>` element is a child element of the `<try>` step that specifies the steps to execute in the event that a typical error occurs while executing the steps of the `<block>` element. It may contain any number of steps that are permitted within the scope of the block containing the `<try>` step.

The `<catch>` element serves both to suppress typical errors of the `<block>` element, and to define typical error recovery actions. Note that it may be empty, in which case it serves only to suppress typical errors.

finally Element

A child element of the `<try>` step that specifies the steps to execute regardless of whether a typical error occurred earlier within the `<try>` or `<catch>` steps; typical errors comprise plan abort or plan timeout, in which case the steps of the `<finally>` element are skipped altogether. It may contain any number of steps that are permitted within the scope of the block containing the `<try>` step.

The `<finally>` element is used primarily to specify steps to clean up and release resources, when the cleanup should be triggered regardless of other typical error conditions.

urlTest Step

The `<urlTest>` is a step used to verify that the contents of a particular URL match an expected pattern. If the desired pattern is not matched, the step fails and execution stops.

Attributes of the urlTest Step

Name	Type	Required	Configurable	Description
delaySecs	positiveInteger	Yes	No	The number of seconds to wait before testing the URL content.
timeoutSecs	positiveInteger	Yes	No	The number of seconds to wait to receive the URL content before failing. This time starts after the delay has completed.
URL	String	Yes	Yes	The URL whose content should be tested. Currently, only HTTP protocol is supported.
pattern	String	Yes	Yes	A glob-style pattern expected to match the content of the URL being tested.

Composite Plan-only Steps

This section lists steps that may only be used within a composite plan. Note that steps contained within a plan may include references to plan variables and parameters.

execSubplan Step

The <execSubplan> step is a step used to execute another plan. The <execSubplan> step may only appear as the child of the <compositeSteps> element.

execSubplan Step attributes

Name	Type	Required	Configurable	Description
------	------	----------	--------------	-------------

planName	entityName	Yes	No	The name of the plan to execute. When this step is run, there must be a corresponding top-level <exectuionPlan> with this name. This name may not reference an inline subplan.
planPath	pathReference	No	No	The path of the plan to execute. If unspecified, the path of the containing plan is assumed.
planVersion	Version	No	No	The version of the plan to execute. If unspecified, the latest version of the named plan is executed.

execSubplan Step child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the called plan. For each parameter in the <paramList> section of the called plan for which no default value is declared, there must be a corresponding argument declared by this <argList>

inlineSubplan Step

The <inlineSubplan> step is a step used to execute a sequential series of steps. This step may only appear as the child of the <compositeSteps> element.

An <inlineSubplan> step is similar to an <execSubplan> step except that the <execSubplan> steps names an external plan to execute, whereas the <inlineSubplan> step directly contains the plan to execute as a child element. The primary difference between an inline subplan and a top-level plan is that inline subplans are not saved as distinct named entities, and therefore may not be externally referenced by <execSubplan> steps, whereas top-level plans are distinct named entities and therefore can be referenced from <execSubplan> steps. Inline subplans are useful in cases where

the subplan content is concise and directly tied to the context and logic of the calling plan, and does not otherwise make sense as a stand-alone plan. In these cases, having all steps in one self-contained unit can facilitate maintenance and readability for the plan.

Unlike top-level plans, inline subplans may not declare parameters. They implicitly inherit the parameters and variables of all enclosing plans. They may declare additional variables local to the inline subplan, which may hide variables and parameters of the enclosing plans. A subplan variable hides the variable of an enclosing plan if both have the same name. In this case, only the value of the variable declared by the innermost subplan is available for use by its steps.

inlineSubplan Step attributes

Name	Type	Required	Configurable	Description
planName	entityName	Yes	No	A name used to identify the inline subplan. This name is used primarily for display purposes, and need not be distinct from names of other plans (inline or top-level).
description	String	No	No	A description of the inline subplan, useful for documentation purposes.

inlineSubplan Element child elements

The <inlineSubplan> element consists of an optional <varList> followed by one additional child element, which is either <simpleSteps> or <compositeSteps>, depending on whether the inline subplan is a simple or composite plan.

Name	How Many?	Description
<varList>	0 or 1	A list of plan variables for use within the inline subplan.
simpleSteps	0 or 1	Contains a list of simple steps. Either this element or the compositeSteps element must be present, but not both.

compositeSteps	0 or 1	Contains a list of composite steps. Either this element or the simpleSteps element must be present, but not both.
----------------	--------	---

Simple Plan-only Steps

This section lists steps that may only be used within a simple plan.

Note – Steps contained within a plan can reference any variables declared by that plan, plus any unhidden variables and parameters of all enclosing plans.

install Step

The <installStep> is a step used to install the resources of a component onto the target host. This causes the steps of the named <installSteps> element of the associated component to be executed. This step may only appear as the child of the <simpleSteps> element.

install Step attributes

Name	Type	Required	Configurable	Description
blockName	entityName	Yes	No	The name of the install block to execute within the target component.

install Step child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the <installSteps> block.
repository component targeter	1	Identifies the component to install.

uninstall Step

The <uninstallStep> is a step used to uninstall the resources of a component currently installed on the target host. This causes the steps of the named <uninstallSteps> element of the associated component to be executed. This step may only appear as the child of the <simpleSteps> element.

uninstall Step attributes

Name	Type	Required	Configurable	Description
blockName	entityName	Yes	No	The name of the uninstall block to execute within the target component.

uninstall Step child elements

Name	How Many?	Description
argList	0 or 1	A list of arguments to pass to the <uninstallSteps> block.
installed component targeter	1	Identifies the component to uninstall

The N1 Service Provisioning System Software Command-Line Interface

This chapter describes the command line interface for the N1 Service Provisioning System software.

The command-line interface (CLI) allows you to submit commands to the Master Server from a shell prompt, a DOS prompt, or a script.

The CLI requires a Command Line Interface Client to be installed on the local machine. If you want to run access the CLI in its interactive mode, using the `cr_cli.j` tool, Jython must be installed on the local machine, as well. For details about system requirements and configurations, please see the *N1 Service Provisioning System 4.1 Installation Guide*.

The Command-Line Execution Client provides a command-line interface to the Master Server. The Command-Line Execution Client enables commands to be executed from the following interfaces.

- the Windows command line or a local shell such as bash
- shell scripts using sh or Perl

To execute these commands, the Command Line Execution Client establishes a secure connection to the Master Server through TCP/IP, SSL, or SSH.

The Command Line Execution Client uses the Jython programming language. Jython is a Java implementation of the high-level, dynamic, object-oriented language Python. You should install Jython on any system on which you plan to install the command line execution client. For more information about Jython and to download Jython, visit <http://www.jython.org>.

The CLI is a suite of tools for accessing the N1 Service Provisioning System software through non-HTML interfaces, such as a Windows prompt, a shell, or a script. CLI commands can be used in scripts to automate provisioning software operations, such as checking in files. They can also be used to access the Master Server from systems that lack a Web browser or an HTTP connection.

The CLI requires a Command Line Execution Client, which can be installed on any server that can make a network connection to the Master Server. The client parses the commands into native objects and sends them to the Master Server. Results of the commands are then translated by the client to a textual format and presented to users.

Login authentication is enforced on most commands in the CLI. Users authenticate themselves by specifying a user name and a password or a session ID with each command.

The Structure of CLI Commands

Whether invoked through `cr_cli` or `cr_cli_j`, all CLI commands use the following format:

```
subsystem.object.command  
arguments
```

For example, the command for adding a host to the database is `hdb.h.add`. This command consists three elements that identify:

- a subsystem, in this case the host database (`hdb`)
- an object to operate on, in this case a host (`h`)
- a command or operation, in this case adding (`add`)

Invoking CLI Commands

There are two tools for invoking the Command Line Interface :

- `cr_cli`, which operates in single-line command mode, executing one command at a time
- `cr_cli_j`, which operates in interactive command mode using a Jython interpreter

Working with the CLI

Authentication by Username and Password or Session ID

All but a few of the commands in the command line interface require authentication by username and password or by session ID.

```
cr_cli -cmd [command] -u [user] -p [password]
```

Note – You do not need a username and password to get help on any of the commands or to list all available commands.

For increased security, place the password in a file and pass the file to the command. You can then use the filesystem to restrict access to the file containing the password. Use the following method to securely enter commands that require authentication. For more information on this technique, see “Passing Arguments to the CLI from a File” on page 150.

Create a file with the password, and pass the file to the command.

```
cr_cli -cmd [command] -u [user] -p exp:[passwordfile]
```

Listing the Commands

To list all the commands available, type:

```
cr_cli -cmd -l
```

To list all the commands with a command prefix, use the wildcard character * after the prefix. For example, the command below lists all the hdb commands:

```
cr_cli -cmd -l hdb.*
```

Getting Help with the CLI

To access the general help for the CLI, enter the command:

```
cr_cli -help
```

To see the help for a specific command, enter the command:

```
cr_cli -cmd <commandname> -h
```

CLI Syntax

CLI syntax includes:

- An initial specification of the CLI tool to be used:
 - `cr_cli` (must be specified at the beginning of each command)
 - `cr_clij` (must be specified when the CLI is first invoked)
- A CLI command
- Any arguments required by the command
- Optionally, additional arguments
- Arguments used for authentication. Either:
 - The `-u user` and `-p password` arguments
 - The `-s sessionID` argument, which specifies the session ID of an authenticated user (see “Passing Data to an Individual CLI Argument” on page 149 for an example using the `-s` option)

The following is an example of the general CLI syntax:

TABLE 4-1 Syntax of a CLI Command.

cr_cli	-cmd hdb.h.lo	-ID	NM:barolo7	-u user -p password
Invokes the single command-line interface.	Name of the command you want to invoke.	An argument for the command	Value to be specified with an argument requiring a value.	User name and password, for authentication. Alternatively, you can specify a session ID.

Selecting the Output Format of a CLI Command

Because you may want to use the output resulting from CLI commands in various ways, you can adjust the format a CLI command uses to return information. There are two types of CLI output formats:

- standard output formats that are available to all CLI commands
- optional, command-specific output formats

Standard Output Formats

The CLI supports the following output formats:

TABLE 4-2 CLI Format Options

Format Option	Result
string	Produces brief output. Use this option for postprocessing
serialized	Produces XML serialized text output. Use this option when you want to pass information to another script.
sink	Discards output. On UNIX systems, directs output to /dev/null.
detail	Formats the output as a table. Includes detailed information.

If you do not specify an output format, the command uses its default format for the output.

To discover the default output format for a given command, refer to the help information for that command by running the command with `-h`:

```
cr_cli -cmd <commandname> -h
```

The default output format will be in asterisks at the bottom of the help.

Command-Specific Output Formats

You can also use the `-h` argument to discover command-specific output formats that are available for a command. Table 4-3 lists command-specific output formats

TABLE 4-3 Command Output Formats for CLI Command Families

Command Format	Description
cdb.detail	Detailed text output of components
cmp.detail	Detailed text output of comparisons
hdb.detail	Detailed text output of host data
net.detail	Detailed text output of network data
net.summary	Summarized output of network data
pdb.detail	Detailed text output of plans
pe.detail	Detailed output from the running of plans
rule.detail	Detailed text output of notification rules

TABLE 4-3 Command Output Formats for CLI Command Families (Continued)

Command Format	Description
udb.deep	Very detailed text output describing users and user groups
udb.detail	Detailed text output describing users and user groups
udb.summary	Summarized text output describing users and user groups

Specifying the Output Format

To specify the output format for a CLI command, use the `-o` argument and list it as the first argument to the command. For example, to select the string output format for the `hdb.h.la` command, you would run the command:

```
cr_cli -cmd hdb.h.la -o string -u [user] -p [password]
```

Note – The output format you specify must appear after the command itself, but before any other arguments, as shown in this example.

Passing Output Data to a File

To pass the data resulting from a command to an external file, you can use the `-of` option. Use `-of` to provide a fully-specified file name into which the N1 Service Provisioning System software can put the data from the command you execute.

For example, to pass the data from creating a host named `myhost` to a file called `hostFile`, type:

```
cr_cli -cmd hdb.h.add -of hostFile -u [user]  
-p [password] -name myhost -tID NM:roxhost
```

The file `hostFile` will contain the host information in detail format (the default formatter for the `hdb.h.add` command).

Note – The `-of` option must occur right after the command name, before any other arguments.

Passing Data to an Individual CLI Argument

To pass data to the CLI from an external file, you can use the `file:` option. For instance, you may want to store the session ID of the session you have just started and then use it to authenticate commands.

To save the session ID to a file called `session`, type:

```
cr_cli -cmd udb.login -u [user] -p [password] > session
```

Then, to pass the session ID to another command, type:

```
cr_cli -cmd hdb.h.la -s file:session
```

Converting the Output Format

To convert the output format of a command that you have passed to a file, you can use the `reformat.util` command.

The `util.reformat` command reads in data from the output file you specify and reformats it using the output format you choose.

For example, if you have created a host and stored the output from the `hdb.h.add` command in serialized format in a file named `hostFile`:

```
cr_cli -cmd hdb.h.add -o serialized -u [user]  
-p [password] -name myhost -tID NM:roxhost > hostFile
```

you can give `util.reformat` the output of the previous command and tell it to use the detail format by passing it `hostFile` using the `file:` option as described in “Passing Data to an Individual CLI Argument” on page 149:

```
cr_cli -cmd util.reformat -u [user] -p [password]  
-o hdb.detail -self file:hostFile
```

which will return:

```
ID: 010010000204-1027365659275-00170-1199101891  
Name: myhost  
Description:  
Virtual: false  
Hidden: false  
Type ID: 010010001024-00000000000000-00001-00000000004  
Attributes:  
<Table is empty>  
Applications:  
<Table is empty>
```

Passing Arguments to the CLI from a File

To pass an argument to the CLI from an input file, you can use the Expand option. To do this, you create a file into which you enter the information you want to pass to the CLI. Each argument and its value must be listed on a separate line. Arguments must be listed in the order in which the command requires them. To pass these arguments to the command, call the command and make the `exp :` argument the first argument.

For example, here are two files (`file1.txt` and `file2.txt`) that can be used in conjunction with the `exp :` option:

```
file1.txt:
hdb.h.1a
-u
[user]
exp: file2.txt
```

```
file2.txt:
-p
[password]
```

To execute them, type:

```
cr_cli -cmd exp:file1.txt
```

The first file will call the second file and execute the command.

Descriptions and Multi-word Names

To specify a description or any other qualifier that includes a space, you must enclose the string in quotes, such as:

```
"This is a multi-word description."
```

You can also escape every space with a backslash (\) if you prefer.

ID Substitution Using NM:

Most of the objects that you create have ID numbers associated with them. This ID number is a unique identifier for an entity, such as a user account or a component, in the repository.

ID numbers are useful because they are unique, but they can be cumbersome to enter as the argument to a command. To use names rather than ID numbers, you can use the tag `NM :` to introduce the name of an object on a command line..

Table 4–6 provides a complete listing of all supported `NM :` mappings.

To use the object name rather than the full ID number, use the following syntax to represent an object ID in any CLI command:

```
NM:<host|user|user group name|host type name|etc.>
```

with no spaces.

You can also use the NM: syntax to identify components and plans. You may optionally specify a version number in addition to the name itself:

```
plan = NM:<plan name>[:<version>]
```

For example:

```
NM:simplePlan  
NM:simplePlan:1.0
```

If you do not specify a version number, the N1 Service Provisioning System software uses the latest version of the component or plan.

Showing and Hiding Objects

Because the objects you store in the repository are deeply interconnected, it is not possible to delete objects. You can choose to hide objects that you are not currently using to keep your work space uncluttered.

Note – Hiding an object does not make it inactive or unusable. For example, if a resource is referenced by a component and the resource is hidden, the next build of the component will still reference the resource. The resource is not deactivated or removed; it is simply hidden from view.

You can specify whether or not to show or hide the following objects.

- Hosts
- Host types
- Host sets
- Host searches
- Users
- Plans
- Components

Working with CLI Commands

The commands available through the CLI can be grouped into several major categories:

TABLE 4-4 Subsystems and CLI Prefixes

Subsystem	CLI Prefix
Categories	cat
Component database	cdb
Configuration generator	cfg
Comparison Engine	cmp
Host Database	hdb
Network Operations	net
Plan Database	pdb
Plan Execution	pe
Rules for Notifications	rule
User Database	udb
Miscellaneous Utilities	util

Required Arguments and Optional Arguments

This command descriptions in this manual and the description of each command available through its help flag use the following notation to describe required arguments and optional arguments for commands.

TABLE 4-5 O/R Notation for Arguments

Notation	What It Means
[O]	Argument is optional
[R]	Argument is required.

TABLE 4-5 O/R Notation for Arguments (Continued)

Notation	What It Means
[O/R]	Argument is usually optional, but may be required in some cases.

For example, the help information for the `hdb.h.add` command lists the following arguments:

```
-u [O/R]: The user username
for authentication: String -p [O/R]: The user password
for authentication: String
-s [O/R]: The session ID for authentication: SessionID
-name: The host name: String
-desc [O]: The host description: String
-tID: The ID of the host type: HostTypeID
-attr [O/R]: The host attributes; required if the host
type requires them: Hashtable
```

In this example, the only required arguments are the host name (`-name`) and the host type ID (`-tID`), in addition to the arguments required to authenticate the user or the session. However, if the host type of the new host requires host attributes to be specified, then the `-attr` argument must be used as well.

Note – As the example above shows, the help information for command does not use the notation `[R]` to identify required arguments. Arguments that are required are simply listed without `[O]` or `[O/R]`. See the `-name` argument above for an example.

The `cr_cli` Tool

To execute one CLI command at a time, use the `cr_cli` tool. This tool invokes the CLI in single-line command mode.

Single-line command mode accepts one command at a time as input. Each command submitted must be complete; the user is not interactively prompted for the next input parameter. Operating in this mode, the Command Line Execution Client does not maintain a command history.

Here's an example of a CLI command executed with the `cr_cli` tool. This command adds a host of type `prodserver` to the host database. The user running this command, `rbarnes`, supplies his password to authenticate himself to the Master Server.

```
cr_cli -cmd hdb.h.add -user
rbarnes -p XyZ572 -name webbl -desc 'web server 1' -tID
prodserver
```

`cr_cli` commands can be stored in a file and be called from a shell script. This is useful for repetitive tasks such as running execution plans, comparisons, or populating hosts.

The `cr_cli` Tool

The interactive command line mode uses the Jython interpreter as its shell. Operating in this mode, the CLI offers you these advantages:

- You don't have to type an entire command on a single line. You can simply enter a command name and then enter the command arguments that `cr_cli` prompts you for.
- You can take advantage of the command history stored by the shell.
- You can call N1 Service Provisioning System software commands from within a Jython script.
- You can create more powerful scripts for more complex, repetitive operations.

Jython Scripts

To call commands from within a Jython script, include the following code at the beginning of the script:

```
from clui import *
app=PyCLUI()
app.execStr(CLI command)
App.close()
```

The assignment `app=PyCLUI()` calls the CLI. The `App.close()` call deletes the instance of this Jython class.

CLI Input Types

Table 4–6 describes the syntax of each type of input argument that may be passed to a CLI command.

TABLE 4-6 Syntax of CLI Input Types

Input Type	Syntax
AppInstance, AppInstanceID	NM:<host name>:(RA LD MS) or [ID:]<ID>
AppTypeCriteria	<comma or pipe separated list of (RA LD MS)> or EMPTY
AttributeCriteriaList	<semicolon separated list of <attribute criteria>> where <attribute criteria> is <attribute name>-(contains equals)-<glob pattern value> where <attribute name> is one of: name or description or hostTypeName or IP or parentHostName or <user defined name>
Boolean	case insensitive, one of: true or yes or t or + or false or no or f or -
Category, CategoryID, Category ID Set	<ID> ID:<ID> NM:<category name>
CompCheckInID	[ID:]<ID>

TABLE 4-6 Syntax of CLI Input Types *(Continued)*

Input Type	Syntax
ComponentTypeRefID	NM:<comp type ref name> [ID:]<ID>
Component, ComponentID	NM:<comp name>[:<build number>] or [ID:]<ID>
ComponentVariableSettings, ComponentVariableSettingsID	NM:<comp name>[:<version>]:<var settings name> or [ID:]<ID>
CriteriaShorthand	any or planStart or planEndNormal or planEndAbnormal or diffStart or diffEndNormal or diffEndAbnormal or system or admin or custom
DifferenceSettings, DifferenceSettingsID	NM:<diff name> or [ID:]<ID>
ExecutionPlan, ExecutionPlanID	NM:<plan name>[:<version>] or [ID:]<ID>
Group, GroupID	NM:<group name> or [ID:]<ID>
Hashtable	<semicolon separated list of <key>[=<value>]> where <value> is one of: <single value> or [<comma separated list of <single values>>] Examples: color=red;emptykey or arraykey=[value1,value2]

TABLE 4-6 Syntax of CLI Input Types (Continued)

Input Type	Syntax
Host, HostID	NM:<host name> or [ID:]<ID>
HostIDArrayArray	<semicolon separated list of <host ID arrays>> where <host ID array> is a <comma separated list of <host or host set IDs>> where <host or host set ID> is one of: HS:<HostSetID> or H:<HostID> Examples: H:NM:host1 or HS:NM:hostSetFoo,H:NM:h1;H:NM:h2
HostIDSet	<comma separated list of <HostIDs>>
HostSearch, HostSearchID	NM:<host search name> or [ID:]<ID>
HostSearchIDSet	<comma separated list of <HostSearchIDs>>
HostSet, HostSetID	NM:<host set name> or [ID:]<ID>
HostSetIDSet	<comma separated list of <HostSetIDs>>
HostType. HostTypeID	NM:<host type name> or [ID:]<ID>
HostTypeVarList	<semicolon separated list of <key>[=<single value>]>
InputStreamWrapper	- {for stdin} or <input filename>
InstalledComponentRef	<comp name>:<comp build>:<install path>
InstalledResourceRef	<resource ID>:<install path>
Level	host or dir or file

TABLE 4-6 Syntax of CLI Input Types (Continued)

Input Type	Syntax
NamedBlockType	One of the three literal constants: install uninstall call
OutputStreamWrapper	one of: <i>[+]<filename> {plus means append}</i> - {for stdout} -: {for stderr}
Permission, PermissionID	NM:userdb.read or NM:userdb.write or NM:host.read or NM:host.write or NM:hostType.read or NM:hostType.write or NM:rule.read or NM:rule.write or NM:component.read or NM:component.write or NM:plan.read or NM:plan.write or NM:plan.run:allhosts or NM:plan.run:<host set name> or NM:diff.read or NM:diff.write or NM:diff.run:allhosts or NM:diff.run:<host set name> or [ID:]<ID>
PhysicalCriteria	<comma or pipe separated list of (VIRT PHYS)> or EMPTY

TABLE 4-6 Syntax of CLI Input Types (Continued)

Input Type	Syntax
ReaderWrapper	- {for stdin} or <input file name>
ResourceID	[ID:]<ID>
RoxAddress	<IP name or address>:<port number>
RuleID, RuleMetaData	NM:<rule name> or [ID:]<ID>
Scope	component or host or hostset
SystemServiceRefID	NM:<system service ref name> [ID:]<ID>
SeverityArray	<empty> or INFO[+] {plus means "or greater"} or WARNING[+] or ERROR[+]
StringArray	<comma separated list of <strings>>
StringArrayArray	<semicolon separated list of <StringArrays>>
Style	mm {for model to model} or mi {for model to directory} or ii {for directory to directory}
TimeInterval	- {for indefinite} or @<Date> {from now to Date} or [<weeks>w][<days>d] [<hours>h][<minutes>m] [<seconds>s][<millisecs>ms] or [<weeks>w][<days>d] [<hours>:][<minutes>]. <seconds>[.<millisecs>]]
User, UserID	NM:<user name> or [ID:]<ID>

TABLE 4-6 Syntax of CLI Input Types *(Continued)*

Input Type	Syntax
WriterWrapper	one of: <i>[+]</i> <filename> <i>{plus means append}</i> - {for stdout} -: {for stderr}

cat: Commands for Managing Categories

This chapter describes the commands that you need to use to manage categories.

You can use categories to classify:

- plans
- components
- resources
- comparisons

The N1 Service Provisioning System software includes a built-in category called `system`, which is automatically applied to internal resources.

Overview of the cat Commands

Table 5–1 summarizes the CLI commands available for managing categories.

TABLE 5–1 Summary of cat Commands

Command	Description
<code>cat.add</code>	Adds a new category.
<code>cat.del</code>	Deletes the specified category
<code>cat.mod</code>	Modifies a category.
<code>cat.la</code>	Lists all the categories that have been defined.

cat.add

This command adds a new category to the repository. The category is given a name and a description.

TABLE 5-2 Arguments and Result for the cat.add Command

Argument/Result		Class	Description
name	[R]	String	The name of the Category
desc	[R]	String	The description of the Category
result		Category	The new Category object

cat.del

This command deletes the specified category. Deleting a category does not delete the objects, such as components and plans, in the category.

Note – The built-in category `system` cannot be deleted.

TABLE 5-3 Argument for the cat.del Command

Argument		Class	Description
ID	[R]	CategoryID	The Category instance ID

cat.mod

This command modifies the specified category. Omitted arguments preserve current values.

Note – The built-in category `system` cannot be modified.

TABLE 5-4 Arguments and Result for the cat.mod Command

Argument/Result		Class	Description
ID	[R]	CategoryID	The Category instance ID
name	[O]	String	The new name of the Category
desc	[O]	String	The new description of the Category
result		Category	The modified Category

cat.la

This command lists all categories defined in the N1 Service Provisioning System software.

TABLE 5-5 Result for the cat.la Command

Result	Class	Description
result	Category	The category instances

cdb: CLI Commands for Managing Components

This chapter describes commands that you need to use to manage components.

- “Overview of the cdb Commands” on page 165
- “cdb.c: Managing Components” on page 166
- “cdb.ic: Managing Installed Components” on page 170
- “cdb.vs: Managing Variable Settings” on page 172
- “cdb.ssr: System Service Ref Commands” on page 175
- “cdb.ctr: Component Type Commands” on page 177

Overview of the cdb Commands

The CLI includes the following sets of commands for managing components.

TABLE 6-1 Sets of Commands for Managing Components

CLI Prefix	Description of Command Set
cdb.c	Commands for managing components (XML models)
cdb.ic	Commands for retrieving information about installed components.
cdb.vs	Commands for managing variable settings objects.
cdb.ssr	Commands for managing system service ref objects.
cdb.ctr	Commands for managing component type ref objects.

This chapter describes all the commands in each of these sets.

cdb.c: Managing Components

The `cdb.c` commands provide general-purpose controls for managing components.

TABLE 6-2 CLI Commands for Managing Components

Command	Description
<code>cdb.c.ci</code>	Checks in non-browsable components and component models.
<code>cdb.c.co</code>	Checks out a component.
<code>cdb.c.la</code>	Lists all versions of all components.
<code>cdb.c.lo</code>	Lists detailed information about a component.
<code>cdb.c.lv</code>	Lists all versions of a component.
<code>cdb.c.mod</code>	Modifies a component.
<code>cdb.c.sc</code>	Applies one or more categories to a component.
<code>cdb.c.sh</code>	Shows or hides a component.
<code>cdb.c.del</code>	Deletes a component

cdb.c.ci

Use the `cdb.c.ci` command to check in certain components. You will need to use this command in the following scenarios.

- You need to check in a non-browsable component (for example: an untyped or container component).
If you need to check in a browsable component, use the `cdb.rsrc.ci` command. For a description of the command and its arguments, see “`cdb.rsrc.ci`” on page 220.
- You need to check in a component model (the XML representation of a component), but not the referenced components or source objects. You might use this functionality when updating control blocks and variable values.
If you want to create new versions of the component and all of its referenced components, use the `cdb.rsrc.ci` command. For a description of the command and its arguments, see “`cdb.rsrc.ci`” on page 220.

TABLE 6-3 Arguments and Result for the cdb.c.ci Command

Argument/Result		Syntax	Description
path	[R]	InputStreamWrapper	The component in XML format
major	[O]	boolean	Whether to checkin as a new major version; default false.
import	[O]	boolean	Whether to import variable settings, default true
hidePrev	[O]	boolean	Whether to hide the previous component, default true.
result		Component	The new component

cdb.c.co

This command checks out a component. It outputs the specified component in XML format.

TABLE 6-4 Argument and Result for the cdb.c.co Command

Arguments/Result		Syntax	Description
comp	[R]	ComponentID	The ID of the component XML to view.
result		Component	The component

cdb.c.la

The command lists all versions of all components.

TABLE 6-5 Arguments and Result for the cdb.c.la Command

Arguments/Result		Syntax	Description
sh	[O]	boolean	Whether hidden components are shown, default false

TABLE 6-5 Arguments and Result for the cdb.c.la Command *(Continued)*

Arguments/Result		Syntax	Description
cat	[O]	CategoryID	Category filter to apply, default “all” (meaning all but system category)
result		SummaryComponentArray	The components

cdb.c.lo

This command lists the details of a specified component.

TABLE 6-6 Arguments and Result for the cdb.c.lo Command

Argument/Result		Syntax	Description
ID	[R]	ComponentID	The ID of the component to view
result		Component	The component

cdb.c.lv

This command lists all the versions of the specified component.

TABLE 6-7 Argument and Result for the cdb.c.lv Command

Argument/Result		Syntax	Description
comp	[R]	Component	The component
result		SummaryComponentArray	All the versions of the component

cdb.c.mod

This command saves a component with new resource versions.

TABLE 6-8 Arguments and Result for the cdb.c.mod Command

Argument/Result		Syntax	Description
comp	[R]	ComponentID	The component

TABLE 6-8 Arguments and Result for the cdb.c.mod Command (Continued)

Argument/Result		Syntax	Description
label	[O]	String	The component label
desc	[O]	String	The component description
rva	[O]	StringArray	The resource versions; use version number, “#” for recommended, “+” for default, and “-” for latest; or omit this argument to use latest for all resources. This argument is only applicable for composite components.
hidePrev	[O]	boolean	Whether to hide the previous build. Default is true.
result		Component	The component

cdb.c.sc

This command associates a component with a set of categories.

TABLE 6-9 Arguments and Result for the cdb.c.sc Command

Argument/Result		Syntax	Description
ID	[R]	ComponentID	The ID of the component to affect
catIDs	[R]	CategoryIDSet	The IDs of the Categories to associate with this component
all	[O]	boolean	Whether to change all versions of the component, default false

cdb.c.sc

This command associates a component with a set of categories.

TABLE 6–10 Arguments and Result for the `cdb.c.sc` Command

Argument/Result		Syntax	Description
ID	[R]	ComponentID	The ID of the component to affect
catIDs	[R]	CategoryIDSet	The IDs of the Categories to associate with this component
all	[O]	boolean	Whether to change all versions of the component, default false

`cdb.c.del`

This command deletes a component.

TABLE 6–11 Arguments for the `cdb.c.del` Command

Argument		Syntax	Description
ID	[R]	ComponentID	The ID of the component to delete
all	[O]	boolean	Whether to delete all versions of the component, default false

`cdb.ic`: Managing Installed Components

The `cdb.ic` commands retrieve information about components that are already installed on hosts.

TABLE 6–12 CLI Commands for Managing Installed Components

Command	Description
<code>cdb.ic.lbc</code>	Lists all the hosts on which a component is installed.

TABLE 6–12 CLI Commands for Managing Installed Components (Continued)

Command	Description
<code>cdb.ic.lbh</code>	Lists all the components installed on a specific host.
<code>cdb.ic.vs.lo</code>	Lists details of the specified generated variable settings object.

`cdb.ic.lbc`

This command lists all the hosts on which a particular component is installed.

TABLE 6–13 Argument and result for the `cdb.ic` Command

Argument/Result		Syntax	Description
comp	[R]	ComponentID	The component ID
result		InstalledComponentBeans	The installed components

`cdb.ic.lbh`

This command lists all the components installed on a particular host.

TABLE 6–14 Argument and Result for the `cdb.ic.lbh` Command

Argument/Result		Syntax	Description
host	[R]	HostID	The host ID
cat	[O]	CategoryID	Category filter to apply; default all
result		InstalledComponentBeans	The installed components

`cdb.ic.vs.lo`

This command lists details of a particular generated variable settings object..

TABLE 6–15 Argument and Result for the cdb.ic.vs.lo Command

Argument/Result		Syntax	Description
ID	[R]	InstalledComponentID	The ID of the installed component whose generated variable settings will be viewed.
result		GeneratedVariableSettings	The generated variable settings

cdb.vs: Managing Variable Settings

The `cdb.vs` commands manage variable settings for components.

TABLE 6–16 CLI Commands for Managing Variable Settings

Command	Description
<code>cdb.vs.add</code>	Adds a new variable settings object.
<code>cdb.vs.del</code>	Deletes a variable settings object.
<code>cdb.vs.imp</code>	Imports a variable settings object from one component into another.
<code>cdb.vs.la</code>	Lists all the variable settings objects associated with a specific component.
<code>cdb.vs.lo</code>	Lists the details of a specific variable settings object.
<code>cdb.vs.mod</code>	Modifies a variable settings object.

cdb.vs.add

This command adds a new variable settings object

TABLE 6–17 Arguments and Result for the cdb.vs.add Command

Argument/Result		Syntax	Result
comp	[R]	ComponentID	The component

TABLE 6–17 Arguments and Result for the `cdb.vs.add` Command (Continued)

Argument/Result		Syntax	Result
name	[R]	String	The new name
vars	[R]	Hashtable	The new override values
result		ComponentVariableSettings	The new component variable settings

cdb.vs.del

This command deletes an existing variable settings object.

TABLE 6–18 Argument for the `cdb.vs.del` Command

Argument		Syntax	Description
vs	[R]	ComponentVariableSettings	The ID of the component variable settings to delete

cdb.vs.imp

This command imports variable settings from one component into another.

TABLE 6–19 Arguments for the `cdb.vs.imp` Command

Argument		Syntax	Description
src	[R]	ComponentID	The component to import variable settings from.
dst	[R]	ComponentID	The component to import variable settings to

cdb.vs.la

This command lists all variable settings objects associated with a particular component.

TABLE 6–20 Argument and Result for the cdb.vs.la Command

Argument/Result		Syntax	Description
comp	[R]	ComponentID	The component.
result		ComponentVariableSettingsArray	The component variable settings.

cdb.vs.lo

This command lists details of a particular variable settings object.

TABLE 6–21 Argument and Result for the cdb.vs.lo Command

Argument/Result		Syntax	Description
vs	[R]	ComponentVariableSettings	The component variable settings to view
result		ComponentVariableSettings	The component variable settings

cdb.vs.mod

This command modifies an existing variable settings object

TABLE 6–22 Arguments and Result for the cdb.vs.mod Command

Argument/Result		Syntax	Description
vs	[R]	ComponentVariableSettings	The component variable settings
name	[O]	String	The new name
vars	[O]	Hashtable	The new override values
result		ComponentVariableSettings	The modified component variable settings

cdb.ssr: System Service Ref Commands

TABLE 6–23 CLI Commands for System Service Ref

Command	Description
cdb.ssr.add	Adds a system service ref
cdb.ssr.mod	Modifies an existing system service ref; omitted arguments preserve current values
cdb.ssr.del	Deletes a system service ref
cdb.ssr.lo	Retrieves a system service ref
cdb.ssr.la	Lists all system service refs

cdb.ssr.add

This command adds a new system service ref.

TABLE 6–24 Argument and Result for the cdb.ssr.add Command

Argument/Result		Syntax	Description
name	[R]	String	The system service ref name
desc	[O]	String	The system service ref description
icn	[R]	String	The name of the referenced installed component
icv	[R]	String	The version of the referenced installed component
icp	[O]	String	The install path of the referenced installed component
result		SystemServiceRef	The new system service ref

cdb.ssr.mod

This command modifies an existing system service ref; omitted arguments preserve current values.

TABLE 6–25 Argument and Result for the cdb.ssr.mod Command

Argument/Result		Syntax	Description
ssr	[R]	SystemServiceRef	The target system service ref
name	[O]	String	The system service ref name
desc	[O]	String	The system service ref description
icn	[O]	String	The name of the referenced installed component
icv	[O]	String	The version of the referenced installed component
icp	[O]	String	The install path of the referenced installed component
result		SystemServiceRef	The modified system service ref

cdb.ssr.del

This command deletes a system service ref..

TABLE 6–26 Argument and Result for the cdb.ssr.del Command

Argument/Result		Syntax	Description
ID	[R]	SystemServiceRefID	The system service ref ID

cdb.ssr.lo

This command retrieves a system service ref.

TABLE 6–27 Argument and Result for the cdb.ssr.lo Command

Argument/Result		Syntax	Description
ID	[R]	SystemServiceRef	The target system service ref
result		SystemServiceRef	The system service ref

cdb.ssr.la

This command lists all system service refs.

TABLE 6–28 Argument and Result for the cdb.ssr.lo Command

Argument/Result		Syntax	Description
result		SystemServiceRefArray	The system service refs

cdb.ctr: Component Type Commands

TABLE 6–29 CLI Commands for System Service Ref

Command	Description
cdb.ctr.add	Adds a new component type ref
cdb.ctr.mod	Modifies an existing component type ref; omitted arguments preserve current values
cdb.ctr.del	Deletes a component type ref
cdb.ctr.lo	Retrieves a component type ref
cdb.ctr.la	Lists all component type refs

cdb.ctr.add

This command adds a new component type ref..

TABLE 6–30 Argument and Result for the cdb.ctr.add Command

Argument/Result		Syntax	Description
name	[R]	String	The component type ref name
desc	[O]	String	The component type ref description
order	[R]	String	The component type ref order
group	[R]	String	The component type ref group
indentLevel	[R]	String	The component type indent level
compref	[R]	String	The name of the component ref within the component type ref
compver	[O]	String	The version of the component ref within the component type ref
result		ComponentTypeRef	The new component type ref

cdb.ctr.mod

Modifies an existing system service ref; omitted arguments preserve current values.

TABLE 6–31 Argument and Result for the cdb.ctr.mod Command

Argument/Result		Syntax	Description
ctr	[R]	ComponentTypeRef	The target component type ref
name	[O]	String	The component type ref name
desc	[O]	String	The component type ref description
order	[O]	String	The component type ref order

TABLE 6-31 Argument and Result for the cdb.ctr.mod Command *(Continued)*

Argument/Result		Syntax	Description
group	[O]	String	The component type ref group
indentLevel	[O]	String	The component type indent level
compref	[O]	String	The name of the component ref within the component type ref
compver	[O]	String	The version of the component ref within the component type ref
result		ComponentTypeRef	The modified component type ref

cdb.ctr.del

This command deletes a component type ref.

TABLE 6-32 Argument and Result for the cdb.ctr.del Command

Argument/Result		Syntax	Description
ID	[R]	ComponentTypeRefID	The component type ref ID

cdb.ctr.lo

This command retrieves a component type ref.

TABLE 6-33 Argument and Result for the cdb.ctr.lo Command

Argument/Result		Syntax	Description
ID	[R]	ComponentTypeRef	The target component type ref
result		ComponentTypeRef	The component type ref

cdb.ctr.la

This command lists all component type refs.

TABLE 6-34 Argument and Result for the cdb.ctr.la Command

Argument/Result	Syntax	Description
result	ComponentTypeRefArray	The component type refs

cfg: CLI Commands for Performing Config-Generation

This chapter describes the command needed to perform config-generation.

Overview of the cfg Command

The `cfg` commands perform config-generation.

TABLE 7-1 Summary of the `cfg` Commands

Command	Description
<code>cfg.gen</code>	Generates configuration for an input.

`cfg.gen`

This command generates configuration for an input.

TABLE 7-2 Arguments for the `cfg.gen` Command

Argument		Syntax	Description
<code>path</code>	[R]	<code>ReaderWrapper</code>	The input source
<code>host</code>	[R]	<code>Host</code>	The target host
<code>comp</code>	[R]	<code>Component</code>	The target component
<code>vs</code>	[O]	<code>ComponentVariableSettings</code>	The target override variable settings

cmp: CLI Commands for Running Comparisons

This chapter describes commands that you need to use to run comparisons.

Overview of the cmp Commands

The cmp commands control comparisons.

TABLE 8-1 Summary of cmp Commands

Command Name	Description
cmp.dj.add	Adds (starts) a new comparison.
cmp.dj.del	Deletes (stops) a new comparison.
cmp.dj.la	Lists running and completed comparisons.
cmp.dj.lo	Retrieves a running comparison to display its status.
cmp.ds.add	Adds comparison settings to the specified comparison.
cmp.ds.la	Lists all comparison settings.
cmp.ds.lo	Retrieves the specified comparison settings.
cmp.ds.sc	Associates a comparison settings with a set of categories.
cmp.ds.mod	Modifies the specified comparison settings.

cmp.dj.add

This command adds (starts) a new comparison.

TABLE 8-2 Argument and Result for the cmp.dj.add Command

Argument/Result		Class	Description
ID	[R]	DifferenceSettings	The comparison settings
result		DifferenceJobID	The ID of the new comparison

cmp.dj.del

This command deletes (stops) a running comparison.

TABLE 8-3 Argument and Result for the cmp.dj.del Command

Argument/Result		Class	Description
ID	[R]	DifferenceJobID	The ID of the running comparison
result		Boolean	True if the comparison was successfully stopped

cmp.dj.la

This command lists running and completed comparisons.

TABLE 8-4 Argument and Result for the cmp.dj.la Command

Argument/Result		Class	Description
max	[O]	Integer	The maximum number of jobs to list; ignored if there are more running comparisons than the given number
result		RunningDiffBeanArray	All the running comparisons and some completed ones

cmp.dj.lo

This command retrieves a running comparison to display its progress.

TABLE 8-5 Arguments for the cmp.dj.lo

Argument/Result		Class	Description
ID	[R]	DifferenceJobID	The ID of the comparison
old	[O]	Boolean	Whether to show old messages; default is false
new	[O]	Boolean	Whether to poll for new messages; default is true

cmp.ds.add

This command adds new comparison settings; some optional arguments may be required for some style/level/scope combinations

TABLE 8-6 Arguments and Result for cmp.ds.add

Argument/Result		Class	Description
name	[R]	String	The name of the comparison settings
desc	[O]	String	The settings" description
style	[R]	Style	The settings" style
level	[R/O]	Level	The settings" level
scope	[R/O]	Scope	The settings" scope
srchID	[R/O]	HostID	The settings" source host ID
dstHID	[R/O]	HostID	The settings" destination host ID
dstHSID	[R/O]	HostSetID	The settings" destination host set ID
srcdir	[R/O]	String	The settings" source directory

TABLE 8-6 Arguments and Result for cmp.ds.add (Continued)

Argument/Result		Class	Description
dstdir	[R/O]	String	The settings' destination directory
ignp	[R/O]	StringArray	The settings' ignore paths
cRef	[R/O]	InstalledComponentRef	The settings' component reference
tout	[R]	TimeInterval	The settings' timeout
fsl	[O]	Boolean	Whether to follow symbolic links or not; default is true
incd	[O]	Boolean	The settings' include subdirectories flag; default is false
srcP	[R/O]	ReaderWrapper	The settings' source prepare mini-plan
dstP	[R/O]	ReaderWrapper	The settings' destination prepare mini-plan
srcC	[R/O]	ReaderWrapper	The settings' source cleanup mini-plan
dstC	[R/O]	ReaderWrapper	The settings' destination cleanup mini-plan
usePlans	[R/O]	Boolean	The settings' flag indicating if mini-plans should be executed; default is true
dstUseSrcP	[R/O]	Boolean	The settings's flag indicating if source prepare mini-plan is used on destination; default is true
dstUseSrcC	[R/O]	Boolean	The settings's flag indicating if source cleanup mini-plan is used on destination; default is true

TABLE 8-6 Arguments and Result for `cmp.ds.add` (Continued)

Argument/Result		Class	Description
result		DifferenceSettings	The new comparison settings

cmp.ds.la

This command lists all comparison settings.

TABLE 8-7 Argument and Result for the `cmp.ds.la` Command

Argument/Result		Class	Description
cat	[O]	CategoryID	Category filter to apply, default “all”
result		DifferenceSettingsArray	All the comparison settings

cmp.ds.lo

This command retrieves the specified comparison settings.

TABLE 8-8 Argument and Result for the `cmp.ds.lo`

Argument/Result		Class	Description
ID	[R]	DifferenceSettingsID	The ID of the comparison settings
result		DifferenceSettings	The comparison settings

cmp.ds.del

This command deleted a comparison settings object.

TABLE 8-9 Argument and Result for the `cmp.ds.del`

Argument/Result		Class	Description
ID	[R]	DifferenceSettingsID	The ID of the comparison settings

cmp.ds.mod

This command adds new comparison settings by using existing settings as a template. Omitted arguments preserve current values when possible.

TABLE 8–10 Argument and Result for the cmp.ds.mod

Argument/Result		Class	Description
ID	[R]	DifferenceSettingsID	The ID of the template comparison settings
desc	[O]	String	The settings' description
style	[O]	Style	The settings' style
level	[R/O]	Level	The settings' level
scope	[R/O]	Scope	The settings' scope
srchID	[R/O]	HostID	The settings' source host ID
dsthID	[R/O]	HostID	The settings' destination host ID
dsthsID	[R/O]	HostSetID	The settings' destination host set ID
srcdir	[R/O]	String	The settings' source directory
dstdir	[R/O]	String	The settings' destination directory
ignp	[R/O]	StringArray	The settings' ignore paths
cRef	[R/O]	InstalledComponentRef	The settings' component reference
tout	[O]	TimeInterval	The settings' timeout
fsl	[O]	Boolean	Whether to follow symbolic links or not; default is true
incd	[O]	Boolean	The settings' include subdirectories flag
srcP	[R/O]	ReaderWrapper	The settings' source prepare mini-plan

TABLE 8–10 Argument and Result for the `cmp.ds.mod` (Continued)

Argument/Result		Class	Description
<code>dstP</code>	[R/O]	ReaderWrapper	The settings' destination prepare mini-plan
<code>srcC</code>	[R/O]	ReaderWrapper	The settings' source cleanup mini-plan
<code>dstC</code>	[R/O]	ReaderWrapper	The settings' destination cleanup mini-plan
<code>usePlans</code>	[R/O]	Boolean	The settings' flag indicating if mini-plans should be executed; default is true
<code>dstUseSrcP</code>	[R/O]	Boolean	The settings's flag indicating if source prepare mini-plan is used on destination; default is true
<code>dstUseSrcC</code>	[R/O]	Boolean	The settings's flag indicating if source cleanup mini-plan is used on destination; default is true
<code>result</code>		DifferenceSettings	The new comparison settings

cmp.ds.sc

This command associates a comparison setting with a set of categories.

TABLE 8–11 Argument and Result for the `cmp.ds.sc` Command

Argument/Result		Syntax	Description
<code>ID</code>	[R]	DifferenceSettingsID	The ID of the comparison setting to affect
<code>catIDs</code>	[R]	CategoryIDSet	The IDs of the categories to associate with this comparison setting

hdb: CLI Commands for Managing Hosts

This chapter describes the commands that you need to manage hosts.

Introduction

The CLI includes the following sets of commands for managing hosts.

TABLE 9–1 Sets of Commands for Managing Hosts

CLI Prefix	Description of Command Set
hdb.a	Commands for managing application instances (Applications such as Remote Agents and Local Distributors).
hdb.h	Commands for managing target hosts.
hdb.hr	Commands for managing host searches.
hdb.hs	Commands for managing host sets.
hdb.ht	Commands for managing host types.

This chapter describes all the commands in each of these sets.

hdb.a: Managing Application Instances

The `hdb.a` family of commands controls instances of the provisioning software's applications, such as Remote Agents and Local Distributors.

TABLE 9-2 Summary of `hdb.a` Commands

Command Name	Description
<code>hdb.a.add</code>	Adds a new application instance
<code>hdb.a.del</code>	Deletes an application instance
<code>hdb.a.la</code>	Lists all application instances
<code>hdb.a.lo</code>	Retrieves information about a specific application instance
<code>hdb.a.mod</code>	Modifies an existing application instance
<code>hdb.a.clear</code>	Clears the resource cache of an application instance

`hdb.a.add`

This command adds a new application instance; it registers a specific configuration of a specific N1 Service Provisioning System software application with the Master Server.

TABLE 9-3 Arguments and Result for the `hdb.a.add` Command

Argument/Result		Syntax	Description
<code>hID</code>	[R]	HostID	The ID of the host to contain the new instance
<code>type</code>	[R]	AppType	The application type (RA LD)
<code>pID</code>	[R]	AppInstanceID	The parent application ID
<code>ip</code>	[R]	String	The application IP address (x.y.z.w)
<code>port</code>	[O]	Integer	The application port (not required for ssh)

TABLE 9-3 Arguments and Result for the hdb.a.add Command (Continued)

Argument/Result		Syntax	Description
conn	[R]	ConnectionType	The connection type (raw ssh ssl)
param	[O]	String	The parameters
result		UIApplicationUpdate	The new application instance and associated warnings

Note – Omitting the `-port` argument or setting the port to 0 on an application instance with connection type `ssh` will cause the default `ssh` port to be used.

hdb.a.del

This command deletes an application instance from the Master Server's internal files. Once deleted, the application instance is no longer recognized by the Master Server.

Note – Deleting an application instance does not delete N1 Service Provisioning System software files on the machine where the application instance was installed.

TABLE 9-4 Argument for the hdb.a.del Command

Argument		Syntax	Description
ID	[R]	AppInstanceID	The application instance ID

hdb.a.la

This command lists all the application instances known to the Master Server.

This command does not accept any arguments.

TABLE 9-5 Result Returned by the hdb.a.la command

Result	Result Syntax	Description
result	AppInstanceArray	The application instances

hdb.a.lo

This command retrieves information about a specified application instance.

TABLE 9-6 Argument and Results of the hdb.a.lo Command

Argument/Result		Syntax	Description
ID	[R]	AppInstanceID	The application instance ID
result		AppInstance	The application instance

hdb.a.mod

This command modifies the specified attributes of an existing application instance. Omitted arguments preserve current values.

TABLE 9-7 Arguments and Result for the hdb.a.mod Command

Argument/Result		Syntax	Description
ID	[R]	AppInstanceID	The application instance ID
pID	[O]	AppInstanceID	The new parent application ID
ip	[O]	String	The new application IP address (x.y.z.w)
port	[O]	Integer	The new application port
conn	[O]	ConnectionType	The new connection type (raw ssh ssl)
param	[O]	String	The new parameters
result		UIAppInstanceUpdate	The modified application instance and associated warnings

Note – Omitting the -port argument or setting the port to 0 on an application instance with connection type `ssh` will cause the default `ssh` port to be used.

hdb.a.clear

This command clears the resource cache of an application instance.

TABLE 9-8 Arguments and Result for the hdb.a.clear Command

Argument/Result		Syntax	Description
ID	[R]	AppInstanceID	The application instance ID

hdb.h: Managing Hosts

The `hdb.h` commands manage target hosts: physical and virtual hosts whose applications you are managing with the N1 Service Provisioning System software.

TABLE 9-9 Summary of hdb.h Commands

Command	Description
<code>hdb.h.add</code>	Adds a new host.
<code>hdb.h.del</code>	Deletes a host
<code>hdb.h.la</code>	Lists all hosts.
<code>hdb.h.lo</code>	Retrieves information about a host.
<code>hdb.h.lq</code>	Queries for matching hosts.
<code>hdb.h.mod</code>	Modifies an existing host.

hdb.h.add

This command adds a new host to the host repository.

TABLE 9-10 Arguments and Results for the hdb.h.add Command

Argument/Result		Syntax	Description
name	[R]	String	The host name
desc	[O]	String	The host description

TABLE 9–10 Arguments and Results for the hdb.h.add Command *(Continued)*

Argument/Result		Syntax	Description
tID	[R]	HostTypeID	The ID of the host type
attr	[O]	Hashtable	The host attributes
hide	[O]	Boolean	Whether the host is hidden, default false
pID	[O]	HostID	The ID of the parent host if virtual, or empty if physical
result		Host	The new host

Note – The `-attr` argument only sets the overrides. Attributes not explicitly included retain their default value.

hdb.h.del

This command deletes a host.

TABLE 9–11 Arguments and Results for the hdb.h.del Command

Argument/Result		Syntax	Description
ID	[R]	HostID	The host ID

hdb.h.la

This command lists all hosts in the repository.

TABLE 9–12 Argument and Result for the hdb.h.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden hosts are shown, default false
result		HostArray	The hosts

hdb.h.lo

This command retrieves information about a host from the repository.

TABLE 9-13 Argument and Result for the hdb.h.lo Command

Argument/Result		Syntax	Description
ID	[R]	HostID	The host ID
result		Host	The host

hdb.h.lq

This command queries for hosts that match either specified criteria or a specified filter.

Note – You must call this command with either a query (with `-query`) or a filter (with `-filt` or `-phys`).

TABLE 9-14 Arguments and Result for the hdb.h.lq Command

Argument/Result		Syntax	Description
query	[O]	AttributeCriteriaList	The query criteria
filt	[O]	AppTypeCriteria	The application type filter
phys	[O]	PhysicalCriteria	A filter to restrict to physical or virtual hosts
sh	[O]	boolean	Whether hidden hosts are shown, default false
result		HostArray	The hosts

hdb.h.mod

This command modifies the specified attributes of an existing host. Omitted arguments preserve current values.

TABLE 9-15 Arguments and Result of the hdb.h.mod Command

Argument/Result		Syntax	Description
ID	[R]	HostID	The host ID
name	[O]	String	The host name

TABLE 9-15 Arguments and Result of the hdb.h.mod Command *(Continued)*

Argument/Result		Syntax	Description
desc	[O]	String	The host description
tlID	[O]	HostTypeID	The ID of the host type
attr	[O]	Hashtable	The host attributes
hide	[O]	Boolean	Whether the host is hidden
pID	[O]	HostID	The ID of the parent host if virtual, or "<null>" if physical
result		Host	The modified host

Note – Use the `-attr` argument to specify the attributes whose values you want to modify. Attributes not explicitly specified retain (or are reset to) their default values.

hdb.hr: Managing Host Searches

Overview

The `hdb.hr` commands manage host searches.

TABLE 9-16 Summary of hdb.hr Commands

Command	Description
<code>hdb.hr.add</code>	Adds a new host search.
<code>hdb.hr.del</code>	Deletes a host search.
<code>hdb.hr.la</code>	Lists all host searches.
<code>hdb.hr.le</code>	Lists all the hosts returned by a host search.
<code>hdb.hr.lo</code>	Retrieves information about a host search.
<code>hdb.hr.mod</code>	Modifies an existing host search.

hdb.hr.add

This command adds a new host search. It enters the host search's name and search criteria in the repository.

TABLE 9-17 Arguments and Result for the hdb.hr.add Command

Argument/Result		Syntax	Description
name	[R]	String	The host search name
desc	[O]	String	The host search description
q	[R/O]	AttributeCriteriaList	The dynamic query; must be specified if filt and phys are not
filt	[R/O]	AppTypeCriteria	A filter to restrict the search to certain types of hosts; must be specified if q and phys are not
phys	[R/O]	PhysicalCriteria	A filter to restrict to physical or virtual hosts; must be specified if q and filt are not
hide	[O]	Boolean	Whether the search is hidden, default false
result		HostSearch	The new host search

hdb.hr.del

This command deletes a host search.

TABLE 9-18 Arguments and Result for the hdb.hr.del Command

Argument/Result		Syntax	Description
ID	[R]	HostSearchID	The host search ID

hdb.hr.la

This command lists all host searches defined in the repository.

TABLE 9–19 Argument and Result for the hdb.hr.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden searches are shown, default false
result		HostSearchArray	The host searches

hdb.hr.le

This command lists all hosts that match the criteria of the specified host search. The lists represents hosts that match these criteria when hdb.hr.le is run.

TABLE 9–20 Arguments and Result for the hdb.hr.le Command

Argument/Result		Syntax	Description
ID	[R]	HostSearchID	The host search ID
sh	[O]	boolean	Whether hidden searches are shown, default false
result		HostArray	The hosts

hdb.hr.lo

This command retrieves information about a specified host search.

TABLE 9–21 Argument and Result for the hr.hr.lo

Argument/Result		Syntax	Description
ID	[R]	HostSearchID	The host search ID
result		HostSearch	The host search

hdb.hr.mod

This command modifies an existing host search. Omitted arguments preserve current values

TABLE 9–22 Arguments and Results for the hdb.hr.mod Command

Argument/Result		Syntax	Description
ID	[R]	HostSearchID	The host search ID
name	[O]	String	The host search name
desc	[O]	String	The host search description
q	[O]	AttributeCriteriaList	The dynamic query
filt	[O]	VarValueCriteria	An optional filter to restrict the search to certain types of hosts
phys	[O]	PhysicalCriteria	A filter to restrict to physical or virtual hosts
sh	[O]	Boolean	Whether the search is hidden
result		HostSearch	The modified host search

hdb.hs: Managing Host Sets

The hdb.hs commands manage host sets.

TABLE 9–23 Summary of hdb.hs Commands

Command	Description
hdb.hs.add	Adds a new host set.
hdb.hs.del	Deletes a host set.
hdb.hs.la	Lists all host sets.
hdb.hs.le	Lists all the hosts contained in a host set.
hdb.hs.lo	Retrieves information about a host set.
hdb.hs.mod	Modifies an existing host set.

hdb.hs.add

This command adds a new host set to the repository.

TABLE 9-24 Arguments and Result for the hdb.hs.add Command

Argument/Result		Syntax	Description
name	[R]	String	The host set name
desc	[O]	String	The host set description
hIDs	[O]	HostIDSet	The IDs of the static member hosts
sIDs	[O]	HostSetIDSet	The IDs of the host subsets
rIDs	[O]	HostSearchIDSet	The IDs of the host searches
sh	[O]	Boolean	Whether the host set is hidden, default false
result		HostSet	The new host set

hdb.hs.del

This command deletes a host set.

TABLE 9-25 Arguments and Result for the hdb.hs.del Command

Argument/Result		Syntax	Description
ID	[R]	HostSetID	The host ID

hdb.hs.la

This command lists all host sets defined in the N1 Service Provisioning System software.

TABLE 9-26 Argument and Result for the hdb.hs.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden host sets are shown, default false
result		HostSetArray	The host sets

hdb.hs.le

This command lists all the hosts contained in the specified host set.

TABLE 9–27 Arguments and Results for the hdb.hs.le Command

Argument		Syntax	Description
ID	[R]	HostSetID	The host set ID
sh	[O]	boolean	Whether hidden host sets are shown, default false
result		HostArray	The hosts

hdb.hs.lo

This command retrieves the specified host set.

TABLE 9–28 Argument and Result for the hdb.hs.lo Command

Argument/Result		Syntax	Description
ID	[R]	HostSetID	The host set ID
result		HostSet	The host set

hdb.hs.mod

This command modifies an existing host set; omitted arguments preserve current values.

TABLE 9–29 Arguments and Result for the hdb.hs.mod Command

Argument/Result		Syntax	Description
ID	[R]	HostSetID	The host set ID
name	[O]	String	The host set name
desc	[O]	String	The host set description
hIDs	[O]	HostIDSet	The IDs of the static member hosts

TABLE 9–29 Arguments and Result for the `hdb.hs.mod` Command *(Continued)*

Argument/Result		Syntax	Description
sIDs	[O]	HostSetIDSet	The IDs of the host subsets
rIDs	[O]	HostSearchIDSet	The IDs of the host searches
hide	[O]	Boolean	Whether the host set is hidden
result		HostSet	The modified host set

hdb.ht: Managing Host Types

Overview

The `hdb.ht` commands manage host types.

TABLE 9–30 Summary of `hdb.ht` Commands

Command	Description
<code>hdb.ht.add</code>	Adds a new host type.
<code>hdb.ht.del</code>	Deletes a host type.
<code>hdb.ht.la</code>	Lists all host types.
<code>hdb.ht.lo</code>	Retrieves information about a host type.
<code>hdb.ht.mod</code>	Modifies an existing host type.

`hdb.ht.add`

This command adds a new host type. It assigns the host type a name and defines the host type's attributes.

TABLE 9-31 Arguments and Result for the hdb.ht.add Command

Argument/Result		Syntax	Description
name	[R]	String	The host type name
desc	[O]	String	The host type description
attr	[R]	HostTypeVarList	The host type attributes
hide	[O]	Boolean	Whether the host type is hidden, default false
result		HostType	The new host type

hdb.ht.del

This command deletes a host type.

TABLE 9-32 Arguments and Result for the hdb.ht.del Command

Argument/Result		Syntax	Description
ID	[R]	HostTypeID	The host type ID

hdb.ht.la

This commands lists all the host types defined, including the default host type, `crhost`.

TABLE 9-33 Argument and Result for the hdb.ht.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden host types are shown, default false
result		HostTypeArray	The host types

hdb.ht.lo

This command retrieves a host type.

TABLE 9–34 Argument and Result for the hdb.ht.lo Command

Argument/Result		Syntax	Description
ID	[R]	HostTypeID	The host type ID
result		HostType	The host type

hdb.ht.mod

This command modifies an existing host type. Omitted arguments preserve current values.

TABLE 9–35 Arguments and Result for the hdb.ht.mod Command

Argument/Result		Syntax	Description
ID	[R]	HostTypeID	The host type ID
name	[O]	String	The host type name
desc	[O]	String	The host type description
attr	[O]	HostTypeVarList	The host type attributes
hide	[O]	Boolean	Whether the host type is hidden
result		HostType	The modified host type

net: CLI Commands for Performing Network Operations

This chapter describes the commands that you need to use to perform network operations.

Overview of the net Commands

The `net` commands perform networking tasks related to N1 Service Provisioning System software applications.

TABLE 10-1 Summary of the net Commands

Command	Description
<code>net.gencfg</code>	Generates the Transport Config file for a N1 Service Provisioning System software application.
<code>net.ping</code>	Checks connectivity to a Remote Agent or Local Distributor by executing the TCP/IP ping command.
<code>net.traceroute</code>	Uses the IP traceroute utility to find the route to a Remote Agent or Distributor

Note – A user session is not needed for the `net` commands.

net.gencfg

This command generates the Transport Config file for an application.

TABLE 10-2 Argument for the net.gencfg Command

Argument/Result		Class	Description
d	[R]	CRAAddress	The Address of the Remote Agent or Distributor to ping in the format [{DNS hostname ipAddress]:port]
result		PingResult	Message indicating the ping success or error details if it fails

net.traceroute

This command uses the IP `traceroute` utility to find the route to a Remote Agent or Distributor

TABLE 10-3 Argument and Result for the net.traceroute Command

Argument/Result		Class	Description
d	[R]	CRAAddress	The Address of the Remote Agent or Distributor to find the route to in the format [{DNS hostname ipAddress]:port]
result		UITraceResult	Message indicating the traceroute success or error details if it fails

pdb: CLI Commands for Managing Plans

This chapter describes the commands that you need to use to manage plans.

Overview of the pdb Commands

You can use the `pdb` commands to manage plans. A plan is an XML document that specifies the operations to be performed on components and hosts.

TABLE 11-1 Summary of the `pdb` Commands

Command	Description
<code>pdb.p.ci</code>	Checks in a new version of a plan.
<code>pdb.p.co</code>	Checks out a plan (outputs a plan in XML).
<code>pdb.p.genplan</code>	Generates and outputs a plan in XML.
<code>pdb.p.la</code>	Lists the latest versions of all plans.
<code>pdb.p.lo</code>	Views a plan.
<code>pdb.p.del</code>	Deletes an execution plan.
<code>pdb.p.lv</code>	Lists all the versions of the specified plan.
<code>pdb.p.sh</code>	Shows or hides a plan.
<code>pdb.p.sc</code>	Associates a plan with a set of categories.

`pdb.p.ci`

Checks in a new version of a plan from XML

TABLE 11–2 Arguments and Result for the pdb.p.ci

Argument/Result		Syntax	Description
path	[R]	InputStreamWrapper	The plan in XML format
major	[O]	boolean	Whether to checkin as a new major version; default false
hidePrev	[O]	boolean	Whether to hide the previous plan, default true
result		ExecutionPlan	The new execution plan

pdb.p.co

This command outputs a plan in XML format to stdout.

TABLE 11–3 Argument for the pdb.p.co Command

Argument		Syntax	Description
plan	[R]	ExecutionPlan	The execution plan

pdb.p.genplan

This command generates and outputs a plan as XML.

TABLE 11–4 Arguments for the pdb.p.genplan Command

Argument		Syntax	Description
ID	[R]	ComponentID	The ID of the component to generate the plan for.
pt	[R]	NamedBlockType[]	List of procedure types (see Converter Summary, below)
pn	[R]	StringArray	List of procedure names to use in this plan.

pdb.p.la

This command lists the latest versions of all plans.

TABLE 11-5 Arguments and Result for the pdb.p.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden plans are shown. Default is false
cat	[O]	CategoryID	Category filter to apply. Default is “all”.
result		ExecutionPlanArray	The plans

pdb.p.lo

This command views an execution plan.

TABLE 11-6 Argument and Result for the pdb.p.lo Command

Argument/Result		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the plan to view
result		ExecutionPlan	The execution plan

pdb.p.del

This command deletes an execution plan.

TABLE 11-7 Argument and Result for the pdb.p.del Command

Argument/Result		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the plan to delete
all	[O]	Boolean	Whether to delete all versions of the plan, default false

pdb.p.lv

This command lists all the versions of the specified plan.

TABLE 11–8 Argument and Result of the pdb.p.lv Command

Argument/Result		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the execution plan
result		ExecutionPlanArray	The execution plans

pdb.p.sh

This command hides or shows (un-hides) a plan.

TABLE 11–9 Arguments for the pdb.p.sh Command

Argument		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the plan to hide or un-hide
hide	[R]	boolean	Whether the plan is set to hidden
all	[O]	boolean	Whether to change all versions of the plan, default false

pdb.p.sc

This command associates a plan with a set of categories.

TABLE 11–10 Arguments for the pdb.p.sc Command

Argument		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the plan to which the categories should be applied.
catIDs	[R]	CategoryIDSet	The IDs of the categories to associate with this plan
all	[O]	boolean	Whether to update the categories for all versions of the plan, default false

pe: CLI Commands for Running Plans

This chapter describes the commands that you need to use to run plans.

Overview of the pe Commands

The pe commands provide tools for running, stopping, and monitoring plans.

TABLE 12-1 Summary of the pe Commands

Command	Description
pe.h.prep	Prepares a set of hosts.
pe.p.en	Displays the output of an execNative step.
pe.p.la	Lists running and completed plans.
pe.p.lo	Lists information about a running or completed plan.
pe.p.del	Deletes the history of a completed plan run.
pe.p.lp	Lists the subplans and targets associated with a plan.
pe.p.run	Runs a plan.
pe.p.stop	Stops a plan that is running
pe.pi.lo	Lists the parameters used to run a plan.

pe.h.prep

This command prepares a set of hosts.

TABLE 12-2 Argument and Result of the pe.h.prep Command

Argument/Result		Syntax	Description
ID	[R]	HostIDSet	A comma-separated list of host IDs
result		TaskID	The ID of the preparation plan run

pe.p.en

This command displays the output of an execNative step.

TABLE 12-3 Arguments and Result for the pe.p.en Command

Argument/Result		Syntax	Description
ID	[R]	StepID	Which step in the plan to display. This can be gathered from the output of pe.p.lo command.
output	[O]	Either "so" or "se"	Show only stdout or stderr. Default: only stdout shown.
result		UIExecNativeOutput	Output of the step.

pe.p.la

This command lists running and completed plans.

TABLE 12-4 Arguments and Result for the pe.p.la Command

Argument/Result		Syntax	Description
max	[O]	Integer	The maximum number of plans to list. This argument is ignored if there are more running plans than the given number.
plan	[O]	String	Restrict results to plans with this name
result		RunningPlanBeanArray	All the running plans and some completed ones

pe.p.lo

This command reports on the status of a plan that is running or that has finished running. It displays the StepID of each step. This command outputs a plan in XML format to stdout.

TABLE 12-5 Arguments for the pe.p.lo Command

Argument		Syntax	Description
ID	[R]	TaskID	The ID of the plan run the view
old	[O]	Boolean	Whether to show old messages; default is false
new	[O]	Boolean	Whether to poll for new messages; default is true

pe.p.del

This command deletes the history of a completed plan run.

TABLE 12-6 Arguments for the pe.p.del Command

Argument		Syntax	Description
ID	[R]	TaskID	The ID of the plan run to delete

pe.p.lp

This command lists the subplans and targets associated with a plan. The named plan is listed first.

TABLE 12-7 Argument and Result for the pe.p.lp Command

Argument/Result		Syntax	Description
ID	[R]	ExecutionPlanID	The ID of the execution plan
result		SubplanPromptArray	Array of subplans and targets

pe.p.run

This command runs a plan. If there are plan prompt variables, it interactively queries the user for responses.

TABLE 12-8 Arguments and Result for the pe.p.run Command

Argument/Result		Syntax	Description
PID	[R]	ExecutionPlanID	The ID of the execution plan to run
tar	[R]	HostIDArrayArray	Individual hosts or host sets on which to run the plan and each of its subplans, ordered according to subplan prompts

TABLE 12-8 Arguments and Result for the `pe.p.run` Command (Continued)

Argument/Result		Syntax	Description
comp	[O/R]	StringArrayArray	Selection of component builds to install as part of plan and subplans. Ordered according to component selectors within subplan prompts. Use build number, or "+" for default, "#" for recommended, and "-" for latest. . Don't specify when there are no build versions.
vs	[O/R]	StringArrayArray	Selection of variable settings to use with each selected component build. Use variable settings name, or "+" for default . Don't specify when there are no variable settings.
po	[O]	Boolean	True if only preflight should be run
pdp	[O]	Boolean	True if detailed preflight should be run; default false.
hr1	[O]	Integer	Limit number of hosts running at the same time; default set by server config file.
pto	[R]	TimeInterval	The maximum plan execution time
nto	[R]	TimeInterval	The maximum native call execution time

TABLE 12-8 Arguments and Result for the pe.p.run Command *(Continued)*

Argument/Result		Syntax	Description
f	[O]	InputStreamWrapper	An InputStreamWrapper object is constructed with the name of a regular file or "-" for standard input. If this argument isn't used, standard input is automatically selected
result		TaskID	The ID of the plan run

pe.p.stop

This command stops a plan that is running.

TABLE 12-9 Argument and Result for the pe.p.stop Command

Argument/Result		Syntax	Description
ID	[R]	TaskID	The ID of the plan to stop running.
result		Boolean	True if the plan was stopped successfully.

pe.pi.lo

This command lists the parameters used to run a plan.

TABLE 12-10 Argument and Result of the pe.pi.lo Command

Argument/Result		Syntax	Description
ID	[R]	TaskID	The ID of the plan run.
result		TaskInfo	The plan run parameters

CLI Commands for Managing Resources

This chapter describes the commands that you need to use to manage resources.

Overview of the `cdb` Commands

You can use the `rsrc` commands to manage resources. The `cdb.rsrc` commands include tools for checking in resources individually or in groups (through a batch files) for listing the versions of resources, determining where resources have been installed, and so on.

Checking in a resource creates a check-in job. A check-in job is a process that lasts until the resource has been fully entered in the repository and assigned a version number. The `cdb.cj` commands enable you to control and monitor check-in jobs.

Table 13–1 summarizes the CLI commands for managing resources.

TABLE 13–1 Summary of the `cdb.rsrc` Commands

Command	Description
<code>cdb.rsrc.ci</code>	Checks in a certain components and their source objects to the repository.
<code>cdb.rsrc.cib</code>	Checks in all the resources listed in a batch file.
<code>cdb.cj.la</code>	Lists check-in jobs.
<code>cdb.cj.lo</code>	Lists the status and details of a check-in job.
<code>cdb.cj.stop</code>	Stops the check-in job.

TABLE 13-1 Summary of the `cdb.rsrc` Commands (Continued)

Command	Description
<code>cdb.rsrc.co</code>	Checks out the specified resource.
<code>cdb.rsrc.rci</code>	Re-checks in a resource.
<code>cdb.rsrc.showopts</code>	Shows the check in options supported by a particular type.

In the descriptions that follow, the term <resource family name> means a resource name which designates the collection of all versions of resources checked in as that name. The term <resource specifier> indicates either a resourceID, or the human-friendly equivalent combination of a resource family name and a version number, separated by a colon (example: “bin/blip/lorp:1.1”).

cdb.rsrc.ci

Use the `cdb.rsrc.ci` command to check in certain components and their source objects. You will need to use this command in the following scenarios.

- You need to check in a browsable component (for example: file or Weblogic EJB)
If you want to check in a non-browsable component, use the `cdb.c.ci` command. For a description of the command and its arguments, see “`cdb.c.ci`” on page 166.
- You need to check in source objects for a simple component.
- You need to check in the referenced components of a browsable, composite component.

Each invocation of the `cdb.rsrc.ci` command is considered a “check-in job,” and can be managed with the CLI commands for managing check-in jobs. For example, to determine which `cdb.rsrc.ci` commands are running, you can run the `cdb.cj.la` command, which lists all the current check-in jobs. You can also pass `compCheckInID` value returned by `cdb.rsrc.ci` as an argument to `cdb.cj.lo` to get status information about a specific check-in job.

TABLE 13-2 Arguments and Result for the `cdb.rsrc.ci` Command

Argument/Result		Syntax	Description
src	[R]	String	The local file/directory being checked in
dst	[R]	String	Which component name to check in as

TABLE 13-2 Arguments and Result for the `cdb.rsrc.ci` Command *(Continued)*

Argument/Result		Syntax	Description
type	[R]	String	The type of the component
platform	[O]	HostSetID	The platform of the component
desc	[O]	String	A description of the component
major	[O]	Boolean	Whether the version increment should be major or minor, default false
config	[O]	Boolean	Whether the component is a config file; the default is false
hidePrev	[O]	Boolean	Whether to hide the latest component; the default is true
includeOwners	[O]	Boolean	Whether to include owner information; the default is true
includeGroups	[O]	Boolean	Whether to include group information; the default is true
addTo	[O]	Boolean	Whether the files being checked in should be added to the existing files to create a new version of the component, instead of completely replacing the existing files
hostID	[O]	HostID	The ID of the local host
redun	[O]	Boolean	Whether redundancy checking should apply; the default is true

TABLE 13-2 Arguments and Result for the cdb.rsrc.ci Command *(Continued)*

Argument/Result		Syntax	Description
exported	[O]	Boolean	Whether the source file was exported; the default is false. If true, the following parameters are not used: includeOwners, includeGroups, and addTo.
exportPath	[O/R]	String	The source path of the (previously exported) component. Required if the exported argument is set to true.
pickerName	[O]	String	The name of the component picker to use (defaults to null for the default picker).
extraOpts	[O]	Hashtable	Names and values for any additional options for the type.
result		CompCheckInID	The ID for this component check in job.

cdb.rsrc.cib

The command is the “check-in batch” command. It checks in all the resources listed in a batch file.

TABLE 13-3 Arguments and Result of the cdb.rsrc.cib

Argument/Result		Syntax	Description
batchfile	[R]	String	The name of the batch file listing the resources to be checked in
haltonerror	[O]	Boolean	When true, first error will halt batch execution, default true

TABLE 13-3 Arguments and Result of the `cdb.rsrc.cib` (Continued)

Argument/Result		Syntax	Description
<code>pwdrelative</code>	[O]	Boolean	When true, relative paths are relative to the user directory; otherwise they are relative to the batchfile location, defaults to false
result		String	Message indicating the operation is complete

Overview of Batch Files

The `rsrc.cib` operates on a batch file that includes a line for each resource that will be checked in. Batch files enable you to check-in large numbers of resources with a single command.

Each line in the batch file corresponds to a single resource on the local machine that will be checked in as a single resource. Each line consists of a series of fields that are separated by the pipe character (`|`). Some fields are optional and may be omitted. If an optional field is omitted but is followed by other fields, the omitted field should be followed by a `|` character, so that `rsrc.cib` can accurately identify each field.

You can include comments in a batch file. Any line that begins with the pound character (`#`) is interpreted as a comment.

Table 13-4 describes the syntax of a line of a batch file.

TABLE 13-4 Syntax of a Line in a Batch File

Content	Optional/Required
The location of the resource on the local machine	Required
The name to be assigned to the resource when checked in	Required
The resource type	Required
The platform the resource is intended for expressed as a <code>HostSetID</code> in the form <code>NM:<platform_name></code> , where <code><platform_name></code> is one of the names listed in Table 13-5.	Optional

TABLE 13-4 Syntax of a Line in a Batch File *(Continued)*

Content	Optional/Required
A description of the resource	Optional
A boolean designation of whether the file is a configuration file	Optional (Default is false)
A boolean designation of whether check-in should be assigned a major version number (e.g., 2.0)	Optional (Default is false)
A boolean designation of whether to hide the previous most recent version of the resource	Optional (Default is true)
A boolean designation of whether to include owner information when storing permissions information	Optional (Default is true)
A boolean designation of whether to include group information when storing permissions information	Optional (Default is true)
A boolean designation of whether the files being checked in should be added to the existing files to create a new version, instead of creating a new version by completely replacing the existing files	Optional (Default is true)
If this resource is being checked in from a host, the host ID of the host from which the resource is being checked in	Optional
A boolean designation of whether redundancy checking should apply	Optional (Default is true)
A boolean designation of whether the source file was exported. If true, the following parameters are not used: includeOwners, includeGroups, and addTo..	Optional (Default is false)
A string identifying the export path of the (previously exported) resource; that is, the type-specific path the resource had before it was exported. .	Required if the exported value is true
The name of the picker to use (optional, defaults to null for the default picker)	
A Hashtable in string form containing extra options supported by the type's exporter. (See cdb.rsrc.showopts.)	

The table below lists the names that you can use in the fourth field of a batch file line to specify a platform for the component.

TABLE 13–5 Names for Platforms

Platform Name	Description
any	Any platform supported by the N1 Service Provisioning System software
AIX - any version	Either IBM AIX 4.3.3 or IBM AIX 5.1
AIX 4.3.3	IBM AIX 4.3.3
AIX 5.1	IBM AIX 5.1
Solaris - any version	Solaris™ 6, Solaris 7, or Solaris 8 releases
Solaris 6	Solaris 6 release
Solaris 7	Solaris 7 release
Solaris 8	Solaris 8 release
Windows 2000 Server	Microsoft Windows 2000 Server
Red Hat Linux	Red Hat Linux 7.2, 7.3, 8.0 Red Hat Advanced Server 2.1

Example of a Line in a Batch File

To check in a local file named `home/etc/myfile` as the component `mypath/mycomponentname` as the resource type file for the platform Solaris 7 with the description “this is my file” and no designation as a configuration file, you would enter the following line in a batch file:

```
/home/myfile|mypath/mycomponentname|file|NM:Solaris 7|this is my file
```

If the file being checked in was a configuration file, you would add a boolean field to the end of the line and the field to `true`. For example:

```
/home/myfile|mypath/mycomponentname|file|NM:Solaris 7|this is my file|true
```

If you wanted to omit a description for the mycomponent, you do change this line to the following (note the adjacent pipe separators):

```
/home/myfile|mypath/mycomponentname|file|NM:Solaris 7||true
```

To check in the component as a major version (e.g., 2.0 as opposed to 1.7), you would add `true` in the boolean field for major version check-ins:

```
/home/myfile|mypath/mycomponentname|file|NM:Solaris 7||true|true
```

If the check in was desired to not hide the previous component, the line above would become (note the additional `false` in the final field):

```
/home/myfile|mypath/mycomponentname|file|NM:Solaris 7| |true  
|true|false
```

Similar format considerations apply to the optional boolean specifying whether to include owner and group information when storing permissions information.

Batch File Syntax

The N1 Service Provisioning System software applies these rules when parsing batch files.

- In fields that are known to be path names, slashes (whether forward or backward) are always translated to accommodate the convention used on the native file system.
- Blank lines are allowed as visual separators of clusters of files.
- Leading or trailing whitespace is not stripped from fields.
- Both absolute and relative paths are allowed in a batchfile. By default, relative paths are interpreted as being relative to the batchfile location; this can be overridden with the `-pwdrelative` flag, in which case relative paths will be interpreted as being relative to the current working directory.

Invocation

Batch check in via text file is invoked via a `cdb.src.cib` command ("cib" = check in batch) of the form

```
cdb.rsrc.cib -batchfile [batchfile location] [-haltonerror true|false] [-pwdrelative true|false]
```

Before checking in any resources, the `cdb.rsrc.cib` command performs a syntax check of the file. Next it verifies the existence of all the local files that are to be checked in. If `cdb.rsrc.cib` detects errors in either of these processes, it reports the errors and halts execution (regardless of the setting of the `-haltonerror` boolean argument described below).

The haltonerror Argument

The command line includes an optional `-haltonerror` argument (false by default) that designates whether or not an error from the check-in of a single file should halt the check-in of subsequent files. This boolean argument applies only to errors encountered after `cdb.rsrc.cib` has performed its preliminary error-checking (described in the section above).

The pwdrelative Argument

The command line includes an optional `-pwdrelative` provision (false by default) that designates whether relative paths in the batch file should be interpreted as being relative to the current working directory (`pwdrelative = true`) or relative to the location of the batchfile (`pwdrelative = false`).

Batch File Processing

Batchfile processing is non-transactional. This means that if batch file processing fails and/or halts before completion, any resources that have been successfully checked in remain checked in, and are not “un”-checked in.

Concurrent batch check ins are not arbitrated. If two different batch check-ins targeting the same set of resources begin to run at the same time, there is no mechanism throttling the processing of one batch file while another completes. Both batch files will be processed in the interleaved manner that results from their proximate timing.

cdb.cj.la

This command lists check-in jobs (components being checked in through the HTML user interface or through a CLI check-in command such as `rsrc.ci`). It lists all the jobs that are currently active, as well as the last 20 jobs that completed.

TABLE 13-6 Result of the `cdb.cj.la` Command

Result	Syntax	Description
result	CompCheckInId[]	The list of check in job IDs

cdb.cj.lo

This command displays the status and details of the specified check-in job. You specify a job by its `compCheckInID`. This value is returned by `cdb.rsrc.ci` when you check in a resource.

TABLE 13-7 Argument and Result of the `cdb.cj.lo` Command

Argument/Result		Syntax	Description
ID	[R]	compCheckInID	The ID of the check in job

TABLE 13-7 Argument and Result of the cdb.cj.lo Command (Continued)

Argument/Result		Syntax	Description
result		compStatus	The check in job that was specified

Note – Because this command requires the ID of the check-in job, it does not support ID NM: notation for its argument. See Appendix A for a detailed description of compCheckInId syntax.

cdb.cj.stop

This command stops the specified check-in job.

TABLE 13-8 Argument and Result of the rsrc.cj.stop

Argument/Result		Syntax	Description
ID	[R]	CompCheckInID	The ID of the check in job
result		CompStatus	The check in job that was specified

Note – Because this command requires the ID of the check-in job, it does not support ID NM: notation for its argument. See Appendix A for a detailed description of CheckInJobID syntax.

cdb.rsrc.co

This command checks out the specified component. It transfers a copy from the repository to the local machine.

TABLE 13-9 Arguments and Result for the cdb.rsrc.co

Argument/Result		Syntax	Description
src	[R]	String	The name of the component to transfer

TABLE 13–9 Arguments and Result for the `cdb.rsrc.co` (Continued)

Argument/Result		Syntax	Description
v	[R]	String	The version of the component
dst	[R]	String	The location where the component is to be placed
result		String	Message indicating the operation is complete

code> cdb.rsrc.rci

This command re-checks in a component. If a check-in job has been interrupted, you can use this command to repeat the check-in without artificially incrementing the version number of the checked-in component.

TABLE 13–10 Argument and Result for the `cdb.rsrc.rci` Command

Argument/Result		Syntax	Description
ID	[R]	ComponentID	The ID of the component to re-check in.
result		CompCheckInID	The ID of the resulting check in job.

rule: CLI Commands for Notifications

This chapter describes the commands that you need to use to manage event notifications.

Overview of the rule Commands

The `rule` commands let you configure email notifications to be sent when specific types of events occur in the N1 Service Provisioning System software.

Table 14-1 summarizes the CLI commands for managing rules for notifications.

TABLE 14-1 Summary of the rule Commands

Command	Description
<code>rule.add</code>	Adds a notification rule.
<code>rule.del</code>	Deletes a notification rule.
<code>rule.la</code>	Lists all notification rules.
<code>rule.lo</code>	Retrieves a specific rule.
<code>rule.mod</code>	Modifies an existing rule.

`rule.add`

This command adds a new rule for sending email notifications.

TABLE 14-2 Arguments and Result for the rule.add Command

Argument/Result		Syntax	Description
n	[R]	String	The rule name
d	[O]	String	The rule description
tf	[O]	CriteriaShorthand	The rule event type filter
mf	[O]	String	The rule message filter
haf	[O]	HostIDArray	The rule hosts filter
hsf	[O]	HostSetID	The rule host set filter
sf	[O]	String	The rule severity filter
ea	[R]	String	The email address to notify
result		RuleMetaData	The new rule

Table 14-3 lists the keywords you can pass with the -tf argument to specify the type of event a rule applies to.

TABLE 14-3 CriteriaShorthand for the -tf Argument

Criteria Shorthand	Event
any	Any event
planStart	A plan starts
planEndAbnormal	A plan ends abnormally
planEndNormal	A plan completes normally
cmpStart	A comparison starts
cmpEndAbnormal	A comparison ends abnormally
cmpEndNormal	A comparison completes normally
system	Any system event
admin	Any administrative event
custom	Any custom events

rule.del

This command deletes the specified rule.

TABLE 14-4 Argument for the rule.del Command

Argument		Syntax	Description
ID	[R]	RuleID	The rule ID

rule.la

This command lists all the rules defined in the N1 Service Provisioning System software.

TABLE 14-5 Result of the rule.la Command

Result	Syntax	Description
result	RuleMetaDataArray	The rules

rule.lo

This command retrieves a rule, which is specified by its ID.

TABLE 14-6 Argument and Result for the rule.lo Command

Argument/Result		Syntax	Description
ID	[R]	RuleID	The rule ID
result		RuleMetaData	The rule

rule.mod

This command modifies the specified attributes of an an existing rule. Omitted arguments preserve current values.

TABLE 14-7 Arguments and Result for the rule.mod Command

Argument/Result		Syntax	Description
ID	[R]	RuleID	The rule ID
n	[O]	String	The rule name
d	[O]	String	The rule description

TABLE 14-7 Arguments and Result for the rule.mod Command *(Continued)*

Argument/Result		Syntax	Description
tf	[O]	CriteriaShorthand	The rule event type filter
mf	[O]	String	The rule message filter
haf	[O]	HostIDArray	The rule hosts filter
hsf	[O]	HostSetID	The rule host set filter
sf	[O]	String	The rule severity filter
ea	[O]	String	The email address to notify
result		RuleMetaData	The modified rule

udb: CLI Commands for Managing Users and Groups

This chapter describes the commands that you need to use to manage users and groups.

Overview of the udb Commands

The CLI includes the following sets of commands for managing users and groups.

TABLE 15–1 Sets of Commands for User Accounts, Groups, and Logins

CLI Prefix	Description of Command Set
udb.g	Commands for managing user groups.
udb.login udb.logout udb.whoami	Commands for managing login sessions
udb.p	Commands for managing permissions
udb.u	Commands for managing user accounts
udb.sv	Commands for managing session variables.

This chapter describes all the commands in each of these sets.

udb.g: Managing User Groups

You can use the `udb.g` commands to define, modify, delete, and list user groups.

TABLE 15-2 Summary of `udb.g` Commands

Command Name	Description
<code>udb.g.add</code>	Adds a new user group
<code>udb.g.del</code>	Deletes a user group
<code>udb.g.la</code>	Lists all the user groups
<code>udb.g.lo</code>	Retrieves information about the specified user group.
<code>udb.g.lp</code>	Lists the permissions granted to the specified group
<code>udb.g.lu</code>	Lists the users who are members of the specified group
<code>udb.g.mod</code>	Modifies an existing user group

`udb.g.add`

This command adds a new group.

TABLE 15-3 Arguments and Result for the `udb.g.add` Command

Argument/Result		Syntax	Description
<code>n</code>	[R]	String	The new group name
<code>d</code>	[O]	String	The new group description
<code>ua</code>	[O]	UserArray	The new group users
<code>pa</code>	[O]	PermissionArray	The new group permissions
<code>pga</code>	[O]	GroupArray	The new group parent groups
<code>cga</code>	[O]	GroupArray	The new group child groups

TABLE 15-3 Arguments and Result for the udb.g.add Command *(Continued)*

Argument/Result	Syntax	Description
result	Group	The new group

udb.g.del

This command deletes the specified group.

Note – Deleting a group does not delete the user accounts in the group. It simply deletes the group as a classification for the user accounts.

TABLE 15-4 Argument for the udb.g.del Command

Argument		Syntax	Description
ID	[R]	GroupID	The group ID

udb.g.la

This command lists all the groups defined in the N1 Service Provisioning System software.

TABLE 15-5 Result for the udb.g.la Command

Result	Syntax	Description
result	GroupArray	The groups

udb.g.lo

This command retrieves the specified group.

TABLE 15-6 Argument and Result for the udb.g.lo Command

Argument/Result		Syntax	Description
ID	[R]	GroupID	The group ID
result		Group	The group

udb.g.lp

This command lists the permissions granted to a group

TABLE 15-7 Argument and Result for the udb.g.lp

Argument/Result		Syntax	Description
ID	[R]	GroupID	The group ID
result		PermissionArray	The permissions

udb.g.lu

This command lists the members of the specified group

TABLE 15-8 Argument and Result for the udb.g.lu Command

Argument/Result		Syntax	Description
ID	[R]	GroupID	The group ID
result		UserArray	The users

udb.g.mod

This command modifies an existing group. Omitted arguments preserve current values

TABLE 15-9 Arguments and Result for the udb.g.mod Command

Argument/Result		Syntax	Description
ID	[R]	GroupID	The group ID
n	[O]	String	The new group name
d	[O]	String	The new group description
ua	[O]	UserArray	The new group users
pa	[O]	PermissionArray	The new group permissions
pga	[O]	GroupArray	The new group parent groups
cga	[O]	GroupArray	The new group child groups
result		Group	The modified group

udb.u: Managing User Accounts

You can use the `udb.u` commands to manage individual user accounts.

TABLE 15–10 Summary of `udb.g` Commands

Command Name	Description
<code>udb.u.add</code>	Adds a new user account
<code>udb.u.cp</code>	Changes the password of the specified user
<code>udb.u.la</code>	Lists all user accounts
<code>udb.u.lo</code>	Retrieves information about the specified user.
<code>udb.u.lp</code>	Lists the permissions granted to the specified user
<code>udb.u.mod</code>	Modifies the specified user account

`udb.u.add`

This command adds a new user.

TABLE 15–11 Arguments and Result for the `udb.u.add` Command

Argument		Syntax	Description
<code>nu</code>	[R]	String	The new user username
<code>np</code>	[R]	String	The new user password
<code>ng</code>	[O]	GroupArray	The new user groups
<code>hide</code>	[O]	boolean	Whether the user is set to hidden, default false
result		User	The new user

`udb.u.cp`

This command changes the password of the specified user.

TABLE 15–12 Arguments for the udb.u.cp Command

Argument		Syntax	Description
un	[R]	String	The username of the user whose password should be changed
op	[R]	String	The old password
np	[R]	String	The new password

udb.u.la

This command lists all user accounts.

TABLE 15–13 Argument and Result for the udb.u.la Command

Argument/Result		Syntax	Description
sh	[O]	boolean	Whether hidden users are shown, default false
result		UserArray	The users

udb.u.lo

The `udb.u.lo` command retrieves the specified user.

TABLE 15–14 Argument and Result for the udb.u.lo Command

Argument/Result		Syntax	Description
ID	[R]	UserID	The user ID
result		User	The user

udb.u.lp

This command lists the permissions granted to a user.

TABLE 15–15 Argument/Result for the udb.u.lp Command

Argument/Result		Syntax	Description
ID	[R]	UserID	The user ID
result		PermissionArray	The permissions

udb.u.mod

This command modifies an existing user; omitted arguments preserve current values

TABLE 15–16 Argument/Result for the udb.u.mod Command

Argument/Result		Syntax	Description
ID	[R]	UserID	The user ID
np	[O]	String	The new user password
ng	[O]	GroupArray	The new user groups
hide	[O]	Boolean	Whether the user is set to hidden
active	[O]	Boolean	Whether the user is set to active
result		User	The modified user

udb.sv: Managing Session Variables

You can use the `udb.sv` commands to manage session variables.

TABLE 15–17 Summary of udb.sv Commands

Command Name	Description
<code>udb.sv.add</code>	Adds a new session variable.
<code>udb.sv.del</code>	Deletes a session variable.
<code>udb.sv.la</code>	Lists all session variables.
<code>udb.sv.lo</code>	Retrieves information about the session variable.

TABLE 15–17 Summary of udb.sv Commands (Continued)

Command Name	Description
udb.sv.mod	Modifies the specified session value.

udb.sv.add

This command adds a new session variable.

Note – If you are logged in to the HTML user interface and you add a session variable through the CLI, the session variable name will display without the value when you refresh the list of variables. To display the new session variable's value, log out of the HTML user interface and log back in.

TABLE 15–18 Arguments and Result for the udb.sv.add Command

Argument		Syntax	Description
name	[R]	String	The new session variable name
secure	[O]	Boolean	Whether or not the value should be displayed; true means no; default false
desc	[O]	String	The new session variable value description
value	[R]	String	The new session variable value for this user. If the value for the variable is an empty string, enter: - value " "
result		SessionVariable	The new session variable

udb.sv.del

This command deletes a session variable.

TABLE 15–19 Arguments for the udb.sv.del Command

Argument		Syntax	Description
name	[R]	String	The name of the session variable to delete

udb.sv.la

This command lists all session variables.

TABLE 15–20 Argument and Result for the udb.sv.la Command

Argument/Result		Syntax	Description
result		SessionVariable	The session variable

udb.sv.lo

This command retrieves the specified session variable

TABLE 15–21 Argument and Result for the udb.sv.lo Command

Argument/Result		Syntax	Description
name	[R]	String	The name of the session variable to show
result		SessionVariable	The session variable

udb.sv.mod

This command modifies a session variable; a password must be set using the -p parameter if variables are to be persisted.

TABLE 15–22 Argument/Result for the udb.sv.mod Command

Argument/Result		Syntax	Description
name	[R]	String	The name of the session variable to modify

TABLE 15–22 Argument/Result for the udb.sv.mod Command *(Continued)*

Argument/Result		Syntax	Description
secure	[O]	String	Whether or not the value should be displayed; true means no; default false
desc	[O]	String	The new session variable description
value	[O]	String	The new session variable value for this user
result		SessionVariable	The new session variable

Authentication Commands

udb.login

Logs in a user and returns a SessionID that can be used for authentication.

TABLE 15–23 Result of the udb.login Command

Result	Syntax	Description
result	SessionID	The session ID

udb.logout

This command logs out the user who runs it.

udb.whoami

This command returns the owner of the current session.

TABLE 15–24 Result of the udb.whoami Command

Result	Syntax	Description
result	UserID	The current user ID

udb.p: Commands for Managing Permissions

The `udb.p` commands enable you to display information about the permissions established in the N1 Service Provisioning System software.

TABLE 15–25 Summary of the udb.p Commands

Command	Description
<code>udb.p.la</code>	Lists all permissions.
<code>udb.p.lo</code>	Retrieves the specified permission.

udb.p.la

This command lists all permissions.

TABLE 15–26 Result for the udb.p.la Command

Result	Syntax	Description
result	PermissionArray	The permissions

udb.p.lo

This command retrieves the specified permission..

TABLE 15–27 Argument and Result for the udb.p.lo Command

Argument/Result		Syntax	Description
ID	[R]	PermissionID	The permission ID
result		Permission	The permission

Configuration Generation

Many applications are almost identical no matter what hosts they are installed and running on, apart from perhaps one configuration file containing host-specific information. These configuration files usually contain values that are specific to the host they are installed on, or are otherwise customized on a per-host or host set basis. For example, you may have different deployment environments such as QA/test, staging/burn in, and production. The configuration values used in each of these environments is probably different. The N1 Service Provisioning System software makes the deployment of each of these environments simpler by allowing you to define different variable settings objects that contain the variable values you want to use in each of those deployment environments.

Adding Substitution Variable Definitions to Components

When you use the N1 Service Provisioning System software to run a plan that deploys an application across multiple hosts, you can use the configuration generation feature to automatically insert the appropriate configuration values for each host. The provisioning software does this by parsing a template of the configuration file that you create.

To do this, you add substitution variable definitions to your components. These can be used, for example, as a way to configure the directory into which an application is installed. Using the provisioning software, you can define and manage different variable settings for application deployments on each of your target hosts.

- Each version of a component can declare its own variable definitions
- Each version of a component has its own variable settings (possibly imported from a previous version)

- Each component can be installed using any of its variable settings

Variable Substitution Values

The values that the provisioning software substitutes when you install an application can be any of the following:

- Value you specify
- Component-specific value such as name or label
- Host-specific value such as IP number or hostname
- Value specific to the target host
- Value associated with a previously installed component
- Value of a session variable

Generation Context

Variable substitution happens when a target step is run on a target host. The step can be in a plan, in a component being installed on the target host, or in a component already installed on the target host. If there is a state associated with the target host and component, it is used to determine the value of a particular substitution variable, and can include:

- Target component - The component to be installed
- Target host - The host upon which the plan is to be run
- Target variable settings source - A collection of name-value pairs that will override the the default values defined in the component
- Local variables - Any variables in the target step itself, such as those you have declared in the enclosing block or plan

Types of Variables Available for Substitution

When you create or edit a component, you can use predefined variable definitions or you can create custom ones to suit your needs.

The syntax to use when defining variable substitutions is:

: [variable]

where variable is the parameter you want to substitute the host-specific value for.

There are four basic kinds of variables you can define within a component:

- Simple substitution variables
- External component substitution variables
- Session variable substitution
- Target substitution variables

Simple Substitution Variables

You use simple substitution to define variables that you then reference later in the same component.

Simple substitution variables use the following syntax:

: [variable]

where variable is one of the following:

- Local variable name - You can use local variable names in steps within a plan and in steps within a component's install, uninstall, and control service blocks.
- Parameter name - You can use parameter names in steps within a plan and in steps within a component's install, uninstall, and control service blocks.
- Predefined component variable name - You can only use predefined component variables in input sources associated with a component, but not for plans.

Predefined component variables are always resolved relative to the actual component that is (or is being) installed, even if the variable reference occurs in a base component. From an accessibility standpoint, predefined component variables are treated as `public` variables.

The predefined component variable names are listed in "Predefined Substitution Variables for Components and Configuration Files" on page 263.

- Dynamic component variable name - You can only use dynamic component variables in input sources associated with a component, but not for plans.

A dynamic component variable name is a variable name declared by the associated component of the target component. In this case, the value to be substituted for the variable reference is either the associated override variables settings value, or else the default value defined by the component. The value of this variable will be expanded before it is substituted.

External Substitution Variables

A variable setting value may include any number of external component substitution references. An external component substitution reference is one of the following:

```
: [primary-component:varname]  
: [primary-component:secondary-component-list:varname]
```

Note – An external component substitution reference does not include any white space.

Above, primary-component is one of the following:

```
explicit-external-component  
secondary-component
```

And secondary-component is one of the following:

```
nested-component  
toplevel-component  
dependee-component
```

The secondary-component-list is a sequence of one or more colon-separated secondary-components.

The varname is either a predefined substitution variable name, or a dynamic substitution variable name declared by the referenced component.

Explicit External Component Expansion

An explicit-external-component explicitly specifies a component that is expected to be installed on a particular host at a particular location, and has the component resolution semantics of the <installedComponent> installed component targeter described in Chapter 2, “Installed Component Targeter” section. It has the following syntax:

```
component-target:explicit-component-reference
```

Above, component-target specifies the host on which the desired component is installed, and is one of the following:

```
component  
component (host-redirect)
```

The first variant specifies the current target host in the generation context. The second variant refers to the ROX host named by host-redirect, as specified in the host redirects section.

Above, explicit-component-reference is a reference to a component already installed on the target host specified by component-target. explicit-component-reference minimally includes the installed component name, but may also include a version and an install path, as follows:

```
full-component-name  
full-component-name#version  
full-component-name@{path}
```

```
full-component-name#version@{path}
```

Above, `full-component-name` is an absolute or relative reference to the desired component. Relative references are expanded relative to the plan or component that contains the `explicit-component-reference`. And `path` is either an absolute path name representing the install path of the desired component, or a substitution variable reference that, when expanded, represents the absolute install path of the desired component. If the former includes the `"}` character, it must be escaped using the sequence `"}}"`.

Nested Component Expansion

A `nested-component` specifies the component that is referenced by a nested component reference declared by the current component, and has the component resolution semantics of the `<nestedRef>` installed component targeter described in Chapter 2, "Installed Component Targeter" section. It has the following syntax:

```
nestedRef:component-reference-name
```

Above, `component-reference-name` is the name of a `<componentRef>` element with "NESTED" `installMode` within the current component.

A `nested-component` may only be used when the current component is a composite component, and may only be used as a primary-component when the external component substitution reference appears within a composite component. When used as a primary-component, it may not be used as the default value of a component variable or override variable setting, as the referenced component will not yet have been installed when the variable is evaluated.

Toplevel Component Expansion

A `toplevel-component` specifies the component that is referenced by a toplevel component reference declared by the current component, and has the component resolution semantics of the `<toplevelRef>` installed component targeter described in Chapter 2, "Installed Component Targeter" section. It has one of the following formats:

```
toplevelRef:component-reference-name  
toplevelRef:component-reference-name@{path}  
toplevelRef(host-redirect):component-reference-name  
toplevelRef(host-redirect):component-reference-name@{path}
```

Above, `component-reference-name` is the name of a `<componentRef>` element with "TOPELVEL" `installMode` within the current component.

Above, `path` and `host-redirect` are the location and host, respectively, at which the referenced component is expected to be installed. `host-redirect` has syntax as specified in the `host redirects` section.

A toplevel-component may only be used when the current component is a composite component, and may only be used as a primary-component when the external component substitution reference appears within a composite component. When used as a primary-component, it may not be used as the default value of a component variable or override variable setting unless the referenced component has already been installed.

Dependee Component Expansion

A dependee-component specifies the component on which the current component depends as a result of a dependency created by a <createDependency> step, and has the component resolution semantics of the <dependee> installed component targeter described in Chapter 2, "Installed Component Targeter" section. It has the following syntax:

```
dependee: dependency-name
```

Above, dependency-name is the name of a dependency created by the current component.

A dependee-component may only be used as a primary-component when the external component substitution reference appears within a component. When used as a primary-component, it may not be used as the default value of a component variable or override variable setting, as the dependency will not yet have been created when the variable is evaluated.

System Service Component Expansion

A system-service-component specifies a system service component that is expected to be installed on the root physical host of the current target, and has the component resolution semantics of the <systemService> installed component targeter described in Chapter 2, "Installed Component Targeter" section. It has the following syntax:

```
systemService:system-service-name
```

System Type Component Expansion

A system-type-component specifies the component derived from a given component type that is most recently installed on a particular host at a particular location, and has the component resolution semantics of the <systemType> installed component targeter described in Chapter 2, "Installed Component Targeter" section. It has the following syntax:

```
systemType:component-type-name
```

```
systemType:component-type-name@{path}
```

`systemType(host-redirect):component-type-name`

`systemType(host-redirect):component-type-name@{path}`

Here, `path` is optional, and if specified, indicates the path in which the desired component is installed. The `host-redirect` is also optional, and indicates the host on which the desired component is installed, as described in the host redirects section.

Container Component Expansion

A container-component specifies the component that contains the current component as a nested reference. It has the following syntax:

```
container
```

A container-component may only be used as a primary-component when the external component substitution reference appears within a component that has been installed as a nested component by another "container" component.

External Component Substitution Reference Resolution

The value of an external component substitution reference is computed by first resolving the component referenced by primary-component. If a secondary-component is used as a primary-component, then the component containing the external component substitution reference serves as the initial current component used when resolving the primary-component. The component resolved by primary-component becomes the new current component. Then, each secondary-component in the secondary-component-list is resolved using the last resolved component as the current component. Finally, we lookup and return the variable named `varname` in the last resolved component.

The value of an external component substitution reference is computed based on the value of the named variable within the referenced component as of the time it was installed. It is an error to reference a variable name that is not declared by the installed component, or to reference a component that is not installed on the target host. It is also an error to reference a variable that is not accessible to the component or plan that declares the external component substitution reference. The variable is accessible if and only if the variable is declared with an accessible access mode, and the component declaring the component is accessible, and each of the primary and secondary components traversed to get to that component are accessible.

Note – External component substitution references are computed at the time that they are encountered during a plan run (as opposed to at the start of the plan run). Therefore, actual installed component being referenced may change based on steps executed earlier in the plan. Furthermore, references to components installed on a host other than the current target host may be affected by other plans running concurrently on the other host. For predictable results, clients should only reference hosts included in the target set of the current plan run (so that they're guaranteed to be locked), and should synchronize any install/uninstall operations on an externally referenced host within the plan using retargeting and/or series execution mode.

Here are some examples of external component substitution references, using explicit-external components:

```
: [component:installApache:label]
: [component:jdk#1.3:classpath]
: [component:webApp#2.4@{/usr/local}:bannerColor]
: [component(/):IIS global settings:install_path]
: [component:webApp@{:[webAppPath]}:bannerColor]
```

In the second to last example, the component is resolved that is installed on the root physical host of the current target host.

In the last line, webAppPath is resolved as the name of a simple substitution variable set in the associated component of the generation context, not a variable defined in the externally referenced component.

Here are some examples of external component substitution references, using nested and toplevel-components:

```
: [nestedRef:ref1:label]
: [toplevelRef:ref2:name]
: [toplevelRef(/):ref3@{/usr/local}:var1]
```

Here is an example of external component substitution references, using dependee-components:

```
: [dependee:app2domain:domainName]
```

Session Variables Substitution Variables

Session variables allow users to enter data, such as their WebLogic credentials, once during a login session. Also, users can securely save the contents of these variables so that they don't need to re-enter them each time they log in. Session variables are integrated at the modeling level using substitution variables. Session variables have the following syntax:

```
session:varname
```

Above, varname is the name of a session variable.

Note – Session variable references of the form session:varname will resolve to the value of the session variable in the current user’s session. An attempt to resolve a session variable that is not defined in the current user’s session results in an error that states that the session variable was not defined. When a failure to generate a session variable is encountered, an attempt will be made to add the session variable with an empty value to the user’s current session variable set.

A session variable may be used anywhere a simple substitution variable reference may be used. However, a session variable cannot be used within the value of another session variable.

Target Substitution Variables

You can use target substitution variables to get values directly from a particular host

Target substitution variables use the following syntaxes:

```
: [target:varname]  
: [target (hostname) :varname]
```

Variable names that begin with target refer to the logical host upon which a plan is currently being run.

The varname parameter you add after the colon represents an attribute value that is specified in the target host’s definition for the application you are installing.

If you specify a (hostname) right after target, the value for varname is retrieved from that particular host, which can be a different host than the host the plan is currently being run on. You can also specify another substitution variable for hostname that, when expanded, resolves to the name of a host.

Host Redirects

Host substitution references and external component substitution references may optionally include a host redirect, in which case the host on which the variable or component lookup will occur is redirected from the current target to a named host. A host redirect has one of the following forms:

parent-ref hostname hostname/parent-ref

Where hostname is either the name of a ROX host, or a substitution variable reference that, when expanded, resolves to the name of a ROX host.

And parent-ref is either "/", meaning the root parent host of the named host (or current target host if hostname is omitted), or a series of one or more slash-separated "..", meaning the nth parent of the named host (or current target host if hostname is omitted) where "n" is the number of ".." that appear. The parent-ref operators are meant primarily for use with virtual hosts, though they may be used with physical hosts as well. Applying either operator to a physical host is a no-op.

Here are some examples of using target substitution variables:

This example specifies to look up variable var1 on host myHost.

```
: [target (myHost) : var1]
```

This example specifies to look up variable var1 on host named by substitution variable hostName

```
: [target ( : [hostName] ) : var1]
```

This example specifies to look up variable var1 on the logical host on which the plan is currently running.

```
: [target : var1]
```

The following are variables predefined for target hosts:

- name – the name of the target host
- description – description of the target host
- hostTypeName – the name of the host type of the target host
- raIP – the IP address of the remote agent on the target host
- raPort – port used by the remote agent on the target host
- raHomeDir - absolute path of the remote agent home directory on the target host
- raDataDir - absolute path of the remote agent data directory on the target host
- raTmp Dir - absolute path of the remote agent temporary directory on the target host
- raConfigDir - absolute path of the remote agent config directory on the target host

Note – In addition to the predefined variables, you can also reference any host attribute declared by the host type of the target host.

The following is an example of the list of variables defined in a component for apache:

```
<varList>
  <var name="domainname" default=": [target :
domainname]" />
  <var name="name" default="apache" />
  <var name="installPath" default="/opt/apache" />
  <var name="execNativeShutdown"
```

```

        default=":[installPath]/bin/apachectlstop" />
<var name="execNativeStartUp"
    default=":[installPath]/bin/apachectlstart" />
</varList>

```

This component specifies that the domain name be dynamically retrieved from the target host, defines the default value of 'name' to be 'apache', and then specifies a variable installPath value to use in the definition of two execNative variables.

Although you cannot use a target substitution variable within a configuration file, you can reference a host-specific value from within a configuration file. This is because the value of the variable you reference may be computed as a host-specific value. For example, you might have the following variable defined within the <varList> section of the component:

```
<var name="pants" value=":[target:pie]">
```

and then reference :[pants] from within a configuration file. When :[pants] is substituted, it will be substituted with the value of the pie variable defined by the target host.

Predefined Host Substitution Shorthands

The shorthands :[/] and :[/] can be used anywhere a host substitution reference may be used.

The shorthand :[/] is expanded to the operating system specific file separator based on the root physical host of the current target host. For example, it is expanded to "/" on UNIX-based operating systems and "\" on Windows.

The shorthand :[:] is expanded to the operating system specific path separator based on the root physical host of the current target host. For example, it is expanded to ":" on UNIX-based operating systems and ";" on Windows.

Using Substitution Variables

You can use simple substitution variables in any input source, including configuration files. You can use external component and target substitution variables in a number of places, including varList/var default values of installSteps, uninstallSteps, controlService and executionPlans.

The following is a complete list of what types of substitution variables can be used in what input sources:

Input source	Type of variable allowed
<component>/<varList>/<var>/default attribute variable settings override value	■ component substitution references ■ host substitution references ■ external component substitution references (including nested and dependee as primary components) ■ session variable references

Input source	Type of variable allowed
<component>/<installList>/<installSteps>/ <paramList>/<param>/default attribute <component>/<uninstallList>/<uninstallSteps>/<paramList>/<param> \ /default attribute <component>/<controlList>/<control>/ <paramList>/<param>/default attribute <component>/<snapshotList>/<snapshot>/ <paramList>/<param>/default attribute	■ component substitution references ■ host substitution references ■ external component substitution references ■ session variable references

Input source	Type of variable allowed
<component>/<installList>/<installSteps>/ <varList>/<var>/default attribute <component>/<uninstallList>/<uninstallSteps>/ < varList>/<var>/default attribute <component>/<controlList>/<control>/ < varList>/<var>/default attribute <component>/<snapshotList>/<snapshot>/<varList>/<var>/default attribute	■ local variable substitution references (previously declared) ■ parameter substitution references ■ component substitution references ■ host substitution references ■ external component substitution references ■ session variable references

Input source	Type of variable allowed
<component>/<installList>/<installSteps>/child steps <component>/<uninstallList>/<uninstallSteps>/child steps <component>/<controlList>/<control>/child steps <component>/<snapshotList>/<snapshot>/<prepare>/child steps <component>/<snapshotList>/<snapshot>/<capture>/child steps <component>/<snapshotList>/<snapshot>/<cleanup>/child steps	■ local variable substitution references ■ parameter substitution references ■ component substitution references

Input source	Type of variable allowed
<code><component>/<resourceRef>/<resource></code> <code><component>/installPath</code>	■ component substitution references (excluding <code>rsrcInstallPath</code>)
Input source	Type of variable allowed
<code><component>/<diff>/<ignore></code> configuration files	component substitution references
Input source	Type of variable allowed
<code><executionPlan>/<paramList>/<param>/default attribute</code>	session variable references
Input source	Type of variable allowed
<code><executionPlan>/<varList>/<var>/default attribute</code> <code><executionPlan>/*/<inlineSubplan>/<varList>/<var>/default attribute</code>	■ session variable references ■ local variable substitution references (previously declared) ■ unhidden local variable substitution references of enclosing plans ■ unhidden parameter substitution references ■ host substitution references (simple plans/subplans only) ■ external component substitution references (simple plans/subplans only, excluding container, nested, toplevel, and dependee as primary component)
Input source	Type of variable allowed
<code><executionPlan>/*/child steps</code>	■ local variable substitution references ■ unhidden local variable substitution references of enclosing plans ■ unhidden parameter substitution references

Input source	Type of variable allowed
hostAttributes	session variable references

The following configuration attributes of a component can include substitution variable definitions:

Parent element	Substitutable attribute
capture/addFile	displayName
component	installPath
installSpec	name
installSpec	path
installSpec	permissions
installSpec	user
installSpec	group
installSpec	deployMode
installSpec	diffDeploy
diff/ignore	path
varList/var	default
paramList/param	default
capture/addFile	path

The following configuration attributes of steps within a component or plan can include substitution variable definitions:

parent element	substitutable attribute
argList/arg	value
execJavaexecJavaexecNative	classNameclasspathdir
execNative/outputFile	name
execNative/errorFile	name
execNative/env	name
execNative/env	value
execNative/exec	cmd

parent element	substitutable attribute
execNative/exec/arg	value
execNative/shell	cmd
execNative/shell	<body>
execNative/successCriteria	outputMatches
execNative/successCriteria	errorMatches
execNative/inputText	<body>
execNative/inputFile	name
execNative	userToRunAs
if/condition/istrue	value
if/condition/equals	value1
if/condition/equals	value2
if/condition/matches	value
if/condition/matches	pattern
processTest	processNamePattern
processTest	user
raise	message
retarget	host
retarget/varList/var	default
sendCustomEvent	message
transform	input
transform	output
transform/stylesheet	<body>
transform/subst	match
transform/subst	replace
transform/source	name
uninstallComponent	installPath
urlTest	url
urlTest	pattern

The following configuration attributes of a plan can include substitution variable definitions:

parent element	substitutable attribute
varList/var	default

The following configuration attributes of installed component targeters can include substitution variable definitions:

parent element	substitutable attribute
installedComponent	installPath
installedComponent	host
systemType	installPath
systemType	host
toplevelRef	installPath
toplevelRef	host

The following configuration attributes of repository component targeters can include substitution variable definitions:

parent element	substitutable attribute
component	host
toplevelRef	host

Escape Sequences Within Files With Substitution Variable Definitions

If for some reason you want to include a sequence in a configuration file that matches the sequence defined for a substitution variable (:[]), you can escape out the sequence by using another [after the sequence. For example, if you wanted to include the text :[pants] in your configuration file, but didn't want N1 Service Provisioning System software to try and substitute a value for pants, you would use :[[pants].

The configuration generator replaces all occurrences of :[[in the file with :[

Note – When the input source for the variables you want to substitute is an attribute of a component component that is written in XML, use the usual rules for XML-based escape sequences:

Character	Escape sequence
"	"
'	'
<	<
>	>
&	&

Predefined Substitution Variables for Components and Configuration Files

N1 Service Provisioning System software supplies some predefined substitution variables for use within components and configurations files:

- `name` – Name of the component
- `description` – Description of the component
- `label` – Label of the component
- `softwareVendor` – Software vendor of the component
- `author` – Author of the component
- `path` - Path of the component

The `path` variable is similar to the `path` attribute of the component, except that it always includes a trailing slash (/). The `path` variable can be directly combined with the `name` variable to create the full name of the component as follows:

`: [path]:[name]`

- `rsrcInstallPath` - The location that where the component resource was installed
The `rsrcInstallPath` variable will be in the file format for the remote agent on which the component is (or is being) installed. This variable is only defined for simple components. Because the value of the `rsrcInstallPath` variable may require other dynamic component variables to be computed, it may not be used within a component variable default value or variable settings override value. For the same reasons, the variable may not be used within the

value of the <component> installPath attribute, or the <resourceRef>/<installSpec> element attributes. These restrictions prevent the possibility of circular references.

Note – Variable names and definitions are case-sensitive.
