APIs for Scripting
# Table of Contents

## Chapter 1

**About This Manual** ................................................................. 1

Please give us feedback ............................................................. 1

## Chapter 2

**P4Ruby** .................................................................................. 3

Introduction ...................................................................................... 3
System Requirements ........................................................................ 3
Installing P4Ruby ............................................................................ 3
Programming with P4Ruby ............................................................. 4
  Connecting to SSL-enabled servers ............................................. 4
P4Ruby classes .................................................................................. 5
  P4 .......................................................................................... 5
  P4Exception .......................................................................... 9
  P4::DepotFile ....................................................................... 9
  P4::Revision ......................................................................... 9
  P4::Integration .................................................................. 10
  P4::Map ............................................................................... 10
  P4::MergeData .................................................................. 11
  P4::Message ....................................................................... 11
  P4::OutputHandler ............................................................ 11
  P4::Progress ...................................................................... 12
  P4::Spec ............................................................................. 12
Class P4 ...................................................................................... 13
Description ....................................................................................... 13
Class Methods ................................................................................. 13
  P4.identify -> aString ....................................................... 13
  P4.new -> aP4 .................................................................. 13
Instance Methods ........................................................................... 14
  p4.api_level= anInteger -> anInteger ........................................... 14
  p4.api_level -> anInteger .................................................. 14
  p4.at_exception_level( lev ) { ... } -> self ................................... 14
  p4.charset= aString -> aString ........................................... 15
  p4.charset -> aString ........................................................ 15
  p4.client= aString -> aString ............................................. 15
  p4.client -> aString .......................................................... 15
  p4.connect -> aBool ......................................................... 15
  p4.connected? -> aBool ...................................................... 16
  p4.cwd= aString -> aString .............................................. 16
  p4.cwd -> aString .............................................................. 16
  p4.delete_<spectype>( [options], name ) -> anArray ................... 16
  p4.disconnect -> true ......................................................... 17
  p4.each_<spectype<( arguments ) -> anArray .......................... 17
  p4.env -> string ................................................................. 18
  p4.errors -> anArray ......................................................... 18
  p4.exception_level= anInteger -> anInteger ......................... 18

---

Ruby

```ruby
# P4Ruby

module P4

  def client
    # Implementation
  end

  def connect
    # Implementation
  end

  def each
    # Implementation
  end

end
```

---

## APIs for Scripting

iii
APIs for Scripting

```plaintext
c# exception_level -> aNumber ................................................................. 18
c# fetch_<spectype>( [name] ) -> aP4::Spec .................................................. 19
c# format_spec( "<spectype">, aHash ) -> aString ............................................ 19
c# format_<spectype> aHash -> aHash .............................................................. 19

c# handler= aHandler -> aHandler ............................................................... 20

c# host= aString -> aString ......................................................................... 20

c# input= ( aString | aHash | anArray ) -> aString | aHash | anArray .................. 20

c# maxlocktime= anInteger -> anInteger ...................................................... 21

c# maxresults= anInteger -> anInteger ......................................................... 21

c# maxresults -> anInteger ......................................................................... 22

c# maxscanrows= anInteger -> anInteger ...................................................... 22

c# maxscanrows -> anInteger ........................................................................ 22

c# messages -> aP4::Message ...................................................................... 22

c# p4config_file -> aString ......................................................................... 23

c# parse_<spectype>( aString ) -> aP4::Spec .................................................. 23

c# parse_spec( "<spectype">, aString ) -> aP4::Spec ......................................... 23

c# password= aString -> aString .................................................................. 23

c# password -> aString ................................................................................ 24

c# port= aString -> aString .......................................................................... 24

c# port -> aString ....................................................................................... 24

c# prog= aString -> aString ......................................................................... 24

c# prog= aString -> aString ........................................................................ 24

c# progress= aProgress -> aProgress .......................................................... 25

c# progress -> aProgress .......................................................................... 25

c# run_<cmd>( arguments ) -> anArray ......................................................... 25

c# run( aCommand, arguments... ) -> anArray .............................................. 25

c# run_filelog( fileSpec ) -> anArray ......................................................... 26

c# run_login( arg... ) -> anArray ............................................................... 27

c# run_password( oldpass, newpass ) -> anArray ......................................... 27

c# run_resolve( args ) [ block ] -> anArray ............................................... 27

c# run_submitt( aHash, [arg... ] ) -> anArray ............................................. 28

c# run_tickets( ) -> anArray ...................................................................... 29

c# save_<spectype>( hashOrString, [options] ) -> anArray ......................... 29

c# server_case_sensitive? -> aBool ............................................................. 29

c# server_level -> anInteger ............................................................... 29

c# server_level -> anInteger .................................................................... 29

c# server_unidecode? -> aBool ..................................................................... 29

c# set_env= ( aString, aString ) -> aBoo1 ................................................... 30

c# streams= -> aBoo1 .................................................................................. 30

c# streams? -> aBoo1 .................................................................................. 30

c# tagged( aBoo1 ) [ block ] ....................................................................... 30

c# tagged= aBoo1 -> aBoo1 ........................................................................ 30

c# tagged? -> aBoo1 ................................................................................... 30

c# ticketfile= aString -> aString ............................................................. 31

c# ticketfile -> aString ............................................................................. 31

c# track= -> aBoo1 .................................................................................... 31

c# track? -> aBoo1 ..................................................................................... 31

c# track_output -> anArray ........................................................................ 31

c# user= aString -> aString ...................................................................... 32
```
APIs for Scripting

- `p4.user` -> aString
- `p4.version` -> aString
- `p4.version= aString -> aString`
- `p4.warnings` -> anArray

Class P4Exception

Class Methods

Instance Methods

Class P4::Map

Class Methods

Instance Methods

Class P4::Integration

Class Methods

Instance Methods

- `df.depot_file` -> aString
- `df.each_revision { |rev| block }` -> revArray
- `df.revisions` -> aArray

Class P4::Revision

Class Methods

Instance Methods

- `rev.action` -> aString
- `rev.change` -> aNumber
- `rev.client` -> aString
- `rev.depot_file` -> aString
- `rev.desc` -> aString
- `rev.digest` -> aString
- `rev.each_integration { |integ| block |} -> integArray
- `rev.filesize` -> aNumber
- `rev.integrations` -> integArray
- `rev.rev` -> aNumber
- `rev.time` -> aTime
- `rev.type` -> aString
- `rev.user` -> aString

Class P4::Integration

Class Methods

Instance Methods

- `integ.how` -> aString
- `integ.file` -> aPath
- `integ.srev` -> aNumber
- `integ.erev` -> aNumber

Class P4::Map

Class Methods

Instance Methods

- `Map.new ( [anArray ] )` -> aMap
- `Map.join (map1, map2)` -> aMap

Instance Methods

- `map.clear` -> true
- `map.count` -> anInteger
- `map.empty?` -> aBool
- `map.insert(aString, [aString ] )` -> aMap
- `map.translate (aString, [aBool ] )` -> aString
- `map.includes? (aString)` -> aBool
APIs for Scripting

map.reverse -> aMap ................................................................. 39
map.lhs -> anArray ................................................................ 39
map.rhs -> anArray ............................................................... 39
map.to_a -> anArray .............................................................. 39

Class P4::MergeData .................................................................. 40
Description ............................................................................... 40
Class Methods ........................................................................ 40
Instance Methods ................................................................... 40
  md.your_name() -> aString .................................................... 40
  md.their_name() -> aString .................................................. 40
  md.base_name() -> aString ................................................... 40
  md.your_path() -> aString .................................................... 40
  md.their_path() -> aString ................................................... 41
  md.base_path() -> aString .................................................... 41
  md.result_path() -> aString ................................................. 41
  md.merge_hint() -> aString .................................................. 41
  md.run_merge() -> aBool ..................................................... 42

Class P4::Message ................................................................... 43
Description ............................................................................... 43
Class methods ......................................................................... 43
Instance methods ................................................................... 43
  message.severity() -> anInteger ........................................... 43
  message.generic() -> anInteger ......................................... 43
  message.msgid() -> anInteger .............................................. 43
  message.to_s() -> aString ................................................... 43
  message.inspect() -> aString .............................................. 43

Class P4::OutputHandler ........................................................... 44
Description ............................................................................... 44
Class Methods ....................................................................... 44
  new P4::MyHandler.new -> aP4::OutputHandler .................. 44
Instance Methods ................................................................... 44
  outputBinary -> int ......................................................... 44
  outputInfo -> int .......................................................... 44
  outputMessage -> int ...................................................... 44
  outputStat -> int .......................................................... 44
  outputText -> int .......................................................... 44

Class P4::Progress .................................................................. 45
Description ............................................................................... 45
Class Methods ....................................................................... 45
  new P4::MyProgress.new -> aP4::Progress .......................... 45
Instance Methods ................................................................... 45
  init -> int ........................................................................ 45
  description -> int ......................................................... 45
  update -> int .................................................................. 45
  total -> int .................................................................... 45
  done -> int .................................................................... 45

Class P4::Spec ......................................................................... 46
Description ............................................................................... 46
Class Methods ....................................................................... 46
  new P4::Spec.new( anArray ) -> aP4::Spec ......................... 46
Instance Methods ................................................................... 46
Chapter 3  

P4Perl ............................................................................................................................. 49

Introduction .................................................................................................................... 49
System Requirements ....................................................................................................... 49
Installing P4Perl ............................................................................................................. 49
Programming with P4Perl ............................................................................................. 50
  Connecting to Perforce over SSL ................................................................................. 50
P4Perl Classes ............................................................................................................... 50
  P4 ................................................................................................................................ 51
  P4::DepotFile .................................................................................................................. 54
  P4::Revision .................................................................................................................... 54
  P4::Integration ................................................................................................................. 55
  P4::Map .......................................................................................................................... 55
  P4::MergeData ................................................................................................................. 56
  P4::Message .................................................................................................................... 57
  P4::OutputHandler ....................................................................................................... 57
  P4::Progress ................................................................................................................... 57
  P4::Resolver .................................................................................................................... 58
  P4::Spec ........................................................................................................................ 58
Class P4 ........................................................................................................................ 59
Description ..................................................................................................................... 59
Class methods ............................................................................................................... 59
  P4::new() -> P4 ........................................................................................................... 59
  P4::Identify() -> string ............................................................................................... 59
  P4::ClearHandler() -> undef ...................................................................................... 60
  P4::Connect() -> bool ................................................................................................. 60
  P4::Disconnect() -> undef .......................................................................................... 60
  P4::ErrorCount() -> integer ....................................................................................... 60
  P4::Errors() -> list ....................................................................................................... 60
  P4::Fetch< spectype>( [name] ) -> hashref ................................................................. 60
  P4::Format< spectype>( hash ) -> string .................................................................... 60
  P4::FormatSpec($spectype,$string) -> string ........................................................... 61
  P4::GetApiLevel() -> integer ..................................................................................... 61
  P4::GetCharset() -> string ......................................................................................... 61
  P4::GetClient() -> string ......................................................................................... 61
  P4::GetCwd() -> string .............................................................................................. 61
  P4::GetEnv( $var ) -> string ...................................................................................... 61
  P4::GetHandler() -> Handler ..................................................................................... 62
  P4::GetHost() -> string .............................................................................................. 62
  P4::GetMaxLockTime( $value ) -> integer ................................................................... 62
  P4::GetMaxResults( $value ) -> integer ................................................................... 62
  P4::GetMaxScanRows( $value ) -> integer ................................................................... 62
  P4::GetPassword() -> string ..................................................................................... 62
  P4::GetPort() -> string .............................................................................................. 62
  P4::GetProg() -> string ............................................................................................ 62
  P4::permited_fields -> anArray ................................................................................ 46
  spec._<fieldname> = aValue .................................................................................... 46
  spec._<fieldname>= aValue -> aValue ..................................................................... 46
  spec.permited_fields -> anArray .............................................................................. 46
APIs for Scripting

P4::GetProgress() -> Progress ................................................................. 62
P4::GetTicketFile([string]) -> string .................................................... 62
P4::GetUser() -> String ........................................................................ 62
P4::GetVersion($string) -> string ........................................................ 62
P4::IsConnected() -> bool ................................................................. 63
P4::IsStreams() -> bool ................................................................. 63
P4::IsTagged() -> bool ..................................................................... 63
P4::IsTrack() -> bool ....................................................................... 63
P4::Iterate<spectype>(arguments) -> object ......................................... 63
P4::Messages() -> list ..................................................................... 63
P4::P4ConfigFile() -> string ............................................................... 64
P4::Parse<spectype>(string) -> hashref ................................................ 64
P4::ParseSpec($spectype,string) -> hashref .......................................... 64
P4::Run<cmd>($arg...) -> list | arrayref ........................................... 64
P4::Run("<cmd>", $arg...) -> list | arrayref ........................................... 65
P4::RunFilelog($sargs, $fileSpec...) -> list | arrayref ......................... 66
P4::RunLogin(...):-> list | arrayref ................................................... 66
P4::RunPassword($oldpass, $newpass) -> list | arrayref ....................... 66
P4::RunResolve($resolver, $args...) -> string ........................................ 66
P4::RunSubmit($arg | $hashref, ...) -> list | arrayref ................................ 66
P4::RunTickets() -> list .................................................................... 67
P4::RunPassword($oldpass, $newpass) -> list | arrayref ....................... 66
P4::Save<spectype>($arg...) -> list | arrayref ........................................ 67
P4::ServerCaseSensitive() -> integer ................................................ 67
P4::ServerLevel() -> integer ............................................................. 67
P4::ServerUnicode() -> integer .......................................................... 68
P4::SetApiLevel($integer) -> undef .................................................... 68
P4::SetCharset($charset) -> undef ..................................................... 68
P4::SetClient($client) -> undef ........................................................ 68
P4::SetCwd($path) -> undef .............................................................. 68
P4::SetEnv($var, $value) -> undef ....................................................... 68
P4::SetHandler(Handler) -> Handler ................................................... 69
P4::SetHost($hostname) -> undef ....................................................... 69
P4::SetInput($string | $hashref | $arrayref) -> undef .......................... 69
P4::SetMaxLockTime($integer) -> undef ............................................. 69
P4::SetMaxResults($integer) -> undef ............................................... 69
P4::SetMaxScanRows($integer) -> undef .......................................... 69
P4::SetPassword($password) -> undef .............................................. 69
P4::SetPort($port) -> undef .............................................................. 69
P4::SetProg($program_name) -> undef ............................................. 70
P4::SetProgress(Progress) -> Progress ............................................ 70
P4::SetProgress(Progress) -> Progress ............................................ 70
P4::SetStream(0 | 1) -> undef .......................................................... 70
P4::SetTicketFile($string) -> string .................................................. 70
P4::SetTrack(0 | 1) -> undef .............................................................. 70
P4::SetUser($username) -> undef ...................................................... 70
P4::SetVersion($version) -> undef .................................................. 70
P4::SetTrackOutput() -> list ............................................................. 71
P4::Tagged(0 | 1 | $coderef) -> undef .................................................. 71
P4::TrackOutput() -> list ................................................................. 71
P4::Warnings() -> list .................................................................... 71
P4::WarningCount() -> integer .......................................................... 71
Class P4::DepotFile ........................................................................ 72
Description .................................................................................. 72
Class Methods ................................................................. 72
Instance Methods ............................................................ 72
$df->DepotFile() -> string ................................................. 71
$df->Revisions() -> array ................................................. 71

Class P4::Revision ............................................................. 73
Description .................................................................................. 73
Class Methods ............................................................................. 73
$rev->Integrations() -> array .................................................. 73
Instance Methods ...................................................................... 73
$rev->Action() -> string ....................................................... 73
$rev->Change() -> integer ..................................................... 73
$rev->Client() -> string ....................................................... 73
$rev->DepotFile() -> string ................................................... 73
$rev->Desc() -> string ......................................................... 73
$rev->Digest() -> string ....................................................... 73
$rev->FileSize() -> string .................................................... 73
$rev->Rev() -> integer ........................................................ 73
$rev->Time() -> string ......................................................... 74
$rev->Type() -> string ........................................................ 74
$rev->User() -> string ........................................................ 74

Class P4::Integration .............................................................. 75
Description .................................................................................. 75
Class Methods ............................................................................. 75
$integ->How() -> string ........................................................ 75
$integ->File() -> string ......................................................... 75
$integ->SRev() -> integer ....................................................... 75
$integ->ERev() -> integer ....................................................... 75

Class P4::Map ................................................................. 76
Description .................................................................................. 76
Class Methods ............................................................................. 76
$map = new P4::Map( [ array ] ) -> aMap .................................. 76
$map->Join( map1, map2 ) -> aMap ........................................... 76
Instance Methods ...................................................................... 76
$map->Clear() -> undef ........................................................ 76
$map->Count() -> integer ....................................................... 76
$map->IsEmpty() -> bool ....................................................... 76
$map->Insert( string ... ) -> undef ........................................... 77
$map->Translate( string, [ bool ] ) -> string .................................. 77
$map->Includes( string ) -> bool .............................................. 77
$map->Reverse() -> aMap ....................................................... 77
$map->Lhs() -> array ........................................................ 77
$map->Rhs() -> array ........................................................ 77
$map->AsString() -> array ................................................... 77

Class P4::MergeData .............................................................. 78
Description .................................................................................. 78
Class Methods ............................................................................. 78
Instance Methods ...................................................................... 78
$md->YourName() -> string .................................................... 78
$md->TheirName() -> string ................................................... 78
$md->BaseName() -> string .................................................... 78
Chapter 4  P4Python ................................................................. 85

Introduction ........................................................................ 85

APIs for Scripting

$md.YourPath() -> string ...................................................... 78
$md.TheirPath() -> string ..................................................... 78
$md.BasePath() -> string ..................................................... 78
$md.ResultPath() -> string .................................................. 78
$md.MergeHint() -> string ................................................... 78
$md.RunMergeTool() -> integer ............................................ 79

Class P4::Message ............................................................. 80
Description ......................................................................... 80
Class methods ..................................................................... 80
Instance methods ............................................................ 80
$message.GetSeverity() -> int ............................................ 80
$message.GetGeneric() -> int ............................................. 80
$message.GetId() -> int ..................................................... 80
$message.GetText() -> int .................................................. 80

Class P4::OutputHandler .................................................... 81
Description ......................................................................... 81
Class Methods ................................................................... 81
Instance Methods ............................................................. 81
$handler.OutputBinary() -> int ......................................... 81
$handler.OutputInfo() -> int ............................................. 81
$handler.OutputMessage() -> int ....................................... 81
$handler.OutputStat() -> int ............................................. 81
$handler.OutputText() -> int ............................................. 81

Class P4::Progress .............................................................. 82
Description ......................................................................... 82
Class Methods ................................................................... 82
Instance Methods ............................................................. 82
$progress.Init() -> int ...................................................... 82
$progress.Description( string, int ) -> int ....................... 82
$progress.Update() -> int .................................................. 82
$progress.Total() -> int .................................................... 82
$progress.Done() -> int .................................................... 82

Class P4::Resolver .............................................................. 83
Description ......................................................................... 83
Class Methods ................................................................... 83
Instance Methods ............................................................. 83
$resolver.Resolve() -> string ............................................. 83

Class P4::Spec ................................................................. 84
Description ......................................................................... 84
Class Methods ................................................................... 84
$spec = new P4::Spec( $fieldMap ) -> array ....................... 84
Instance Methods ............................................................. 84
$spec-><fieldname> -> string ........................................... 84
$spec-><fieldname>( $string ) -> string ............................. 84
$spec->PermittedFields() -> array ..................................... 84
## APIs for Scripting

System Requirements ................................................................. 85
Installing P4Python ........................................................................ 85
Programming with P4Python ....................................................... 86
    Submitting a Changelist .......................................................... 87
    Logging into Perforce using ticket-based authentication .......... 87
    Connecting to Perforce over SSL ............................................. 88
    Changing your password ....................................................... 88
    Timestamp conversion ......................................................... 88
    Working with comments in specs .......................................... 89

P4Python Classes ........................................................................ 89
P4 .................................................................................................. 90
P4.P4Exception .......................................................................... 93
P4.DepotFile ................................................................................ 93
P4.Revision ............................................................................... 94
P4.Integration ............................................................................ 94
P4.Map ...................................................................................... 94
P4.MergeData ............................................................................ 95
P4.Message ............................................................................... 96
P4.OutputHandler ...................................................................... 96
P4.Progress ............................................................................... 96
P4.Resolver .............................................................................. 97
P4.Spec ..................................................................................... 97

Class P4 ..................................................................................... 98
Description ................................................................................ 98

Instance Attributes .................................................................... 98
p4.api_level -> int ................................................................. 98
p4.charset -> string ................................................................. 99
p4.client -> string ................................................................. 99
p4.cwd -> string ..................................................................... 99
p4.disable_tmp_cleanup -> string .......................................... 99
p4.encoding -> string ............................................................. 100
p4.errors -> list (read-only) .................................................. 100
p4.exception_level -> int ....................................................... 100
p4.handler -> handler ........................................................... 101
p4.host -> string .................................................................... 101
p4.ignore_file -> string ........................................................ 101
p4.input -> string | dict | list .................................................. 101
p4.iterate_<spectype>( arguments ) -> P4.Spec ....................... 102
p4.maxlocktime -> int ............................................................. 102
p4.maxresults -> int ............................................................... 102
p4.maxscanrows -> int ............................................................ 102
p4.messages -> list (read-only) .............................................. 102
p4.p4config_file -> string (read-only) ................................... 103
p4.password -> string ............................................................ 103
p4.port -> string .................................................................... 103
p4.prog -> string .................................................................... 103
p4.progress -> progress ........................................................ 104
p4.server_case_insensitive -> boolean ................................... 104
p4.server_level -> int (read-only) ......................................... 104
p4.server_unicode -> boolean .................................................. 104
p4.streams -> int .................................................................... 104
APIs for Scripting

Class P4.Revision

Description .............................................................................................................. 120
Instance Attributes .................................................................................................... 120
Instance Methods ...................................................................................................... 119
Class Methods .......................................................................................................... 119

Class P4_DepotFile

Description ............................................................................................................... 120
Instance Attributes .................................................................................................... 120
Instance Methods ...................................................................................................... 119
Class Methods .......................................................................................................... 119

Class Attributes ..................................................................................................... 118

Class P4.Revision

Description .............................................................................................................. 120
Instance Attributes .................................................................................................... 120
Instance Methods ...................................................................................................... 119
Class Methods .......................................................................................................... 119

Class P4_DepotFile

Description ............................................................................................................... 120
Instance Attributes .................................................................................................... 120
Instance Methods ...................................................................................................... 119
Class Methods .......................................................................................................... 119

Class P4.Revision

Description .............................................................................................................. 120
Instance Attributes .................................................................................................... 120
Instance Methods ...................................................................................................... 119
Class Methods .......................................................................................................... 119
APIs for Scripting

rev.action -> string ................................................................. 120
rev.change -> int ................................................................. 120
rev.client -> string ............................................................ 120
rev.depotFile -> string ........................................................ 120
rev.desc -> string ............................................................... 120
rev.digest -> string ............................................................ 120
rev.fileSize -> string .......................................................... 120
rev.integrations -> list ......................................................... 120
rev.rev -> int ................................................................. 120
rev.time -> datetime .......................................................... 121
rev.type -> string ............................................................. 121
rev.user -> string ............................................................. 121

Class Methods ................................................................................ 121
Instance Methods ........................................................................ 121

Class P4.Integration ........................................................................... 122
Description ..................................................................................... 122
Instance Attributes ......................................................................... 122
integ.how -> string ...................................................................... 122
integ.file -> string ...................................................................... 122
integ.srev -> int ........................................................................... 122
integ.erev -> int ........................................................................... 122

Class Methods ................................................................................ 122
Instance Methods ........................................................................ 122

Class P4.Map ...................................................................................... 123
Description ..................................................................................... 123
Instance Attributes ......................................................................... 123

Class Methods ................................................................................ 123
P4.Map.join ( map1, map2 ) -> P4.Map .......................................... 123
Instance Methods ........................................................................ 123
map.clear() .................................................................................. 123
map.count() -> int ....................................................................... 123
map.is_empty() -> boolean ........................................................ 124
map.insert( string ... ) ............................................................... 124
map.translate ( string, [ boolean ] ) -> string ................................ 124
map.includes( string ) -> boolean .............................................. 124
map.reverse() -> P4.Map .......................................................... 124
map.lhs() -> list .......................................................................... 124
map.rhs() -> list .......................................................................... 124
map.as_array() -> list ............................................................... 125

Class P4.MergeData ............................................................................ 126
Description ..................................................................................... 126
Instance Attributes ......................................................................... 126
md.your_name -> string ........................................................... 126
md.their_name -> string ............................................................ 126
md.base_name -> string ............................................................ 126
md.your_path -> string ............................................................ 126
md.their_path -> string ............................................................. 126
md.base_path -> string ............................................................. 126
md.result_path -> string ........................................................... 126
md.merge_hint -> string ............................................................ 126
APIs for Scripting

Instance Methods 126
- md.run_merge() -> boolean 126

Class P4.Message 127
Description 127
Class Methods 127
Instance Attributes 127
- message.severity -> int 127
- message.generic -> int 127
- message.msgid -> int 127

Class P4.Progress 129
Description 129
Class Methods 129
Instance Methods 129
- progress.init() -> int 129
- progress.update() -> int 129
- progress.setDescription( string, int ) -> int 129
- progress.done() -> int 129

Class P4.Resolver 130
Description 130
Instance Attributes 130
Class Methods 130
Instance Methods 130
- resolver.resolve( self, mergeData ) -> string 130

Class P4.Spec 131
Description 131
Instance Attributes 131
- spec._<fieldname> -> string 131
- spec.comment -> dict 131
- spec.permitted_fields -> dict 131

Class Methods 131
- P4.Spec.new( dict ) -> P4.Spec 131

Instance Methods 131

Chapter 5 P4PHP 133

Introduction 133
APIs for Scripting

System Requirements ................................................................. 133
Installing P4PHP ........................................................................ 133
Programming with P4PHP ........................................................ 133
  Submitting a Changelist ......................................................... 134
  Logging into Perforce using ticket-based authentication .......... 135
  Connecting to Perforce over SSL .......................................... 135
  Changing your password ......................................................... 135

P4PHP Classes .............................................................................. 136
  P4 ......................................................................................... 136
  P4_Exception ........................................................................ 139
  P4_DepotFile ......................................................................... 139
  P4_Revision .......................................................................... 139
  P4_Integration ....................................................................... 140
  P4_Map ................................................................................ 140
  P4_MergeData ....................................................................... 141
  P4_OutputHandlerAbstract .................................................. 141
  P4_Resolver .......................................................................... 142

Class P4 ....................................................................................... 143
Description ................................................................................ 143
Properties .................................................................................. 143
  P4::api_level -> int ............................................................. 143
  P4::charset -> string ........................................................... 144
  P4::client -> string ............................................................. 144
  P4::cwd -> string ............................................................... 144
  P4::errors -> array (read-only) .......................................... 145
  P4::exception_level -> int ................................................ 145
  P4::expand_sequences -> bool ........................................... 145
  P4::handler -> handler ...................................................... 146
  P4::host -> string .............................................................. 146
  P4::input -> string | array ................................................ 146
  P4::maxlocktime -> int ...................................................... 147
  P4::maxresults -> int ........................................................ 147
  P4::maxscanrows -> int ..................................................... 147
  P4::p4config_file -> string (read-only) .............................. 147
  P4::password -> string ....................................................... 147
  P4::port -> string ............................................................. 148
  P4::prog -> string ............................................................. 148
  P4::server_level -> int (read-only) ..................................... 149
  P4::streams -> bool ........................................................... 149
  P4::tagged -> bool ............................................................ 149
  P4::ticket_file -> string .................................................... 149
  P4::user -> string ............................................................. 149
  P4::version -> string ........................................................ 150
  P4::warnings -> array (read-only) ..................................... 150

Constructor .............................................................................. 150
  P4::__construct ................................................................. 150

Static Methods ........................................................................ 151
  P4::identify() -> string ..................................................... 151

Instance Methods ..................................................................... 151
  P4::connect() -> bool ......................................................... 151
  P4::connected() -> bool ..................................................... 151
APIs for Scripting

P4::delete_<spectype>( [ options ], name ) -> array .............................................. 152
P4::disconnect() -> void ........................................................................................ 152
P4::env( var ) -> string .......................................................................................... 153
P4::fetch_<spectype>() -> array .............................................................................. 153
P4::format_spec( "<spectype>\"", array ) -> string .............................................. 153
P4::format_<spectype>( array ) -> string .................................................................. 154
P4::parse_spec( "<spectype>\"", string ) -> array ..................................................... 154
P4::parse_<spectype>( string ) -> array .................................................................... 154
P4::run( <cmd>, [arg, ...] ) -> mixed ....................................................................... 154
P4::run_<cmd>() -> mixed ....................................................................................... 156
P4::run_filelog( <fileSpec> ) -> array ................................................................... 157
P4::run_login( arg... ) -> array .............................................................................. 157
P4::run_password( oldpass, newpass ) -> array ...................................................... 157
P4::run_resolve( [<resolver>], [arg...]) -> array .................................................... 158
P4::run_submit( [ array ], [ arg... ] ) -> array ....................................................... 159
P4::save_<spectype>()> .......................................................................................... 159

Class P4_Exception ................................................................................................. 160
  Description ............................................................................................................. 160
  Class Attributes .................................................................................................... 160
  Static Methods ..................................................................................................... 160

Class P4_DepotFile .................................................................................................... 161
  Description ............................................................................................................. 161
  Properties .............................................................................................................. 161
  $df->depotFile -> string ....................................................................................... 161
  $df->revisions -> array ....................................................................................... 161
  Static Methods ..................................................................................................... 161
  Instance Methods ................................................................................................ 161

Class P4_Revision .................................................................................................... 162
  Description ............................................................................................................. 162
  Properties .............................................................................................................. 162
  $rev->action -> string ......................................................................................... 162
  $rev->change -> long ............................................................................................ 162
  $rev->client -> string ........................................................................................... 162
  $rev->depotFile -> string ..................................................................................... 162
  $rev->desc -> string ............................................................................................. 162
  $rev->digest -> string .......................................................................................... 162
  $rev->fileSize -> long .......................................................................................... 162
  $rev->integrations -> array ................................................................................. 162
  $rev->rev -> long .................................................................................................. 162
  $rev->time -> string .............................................................................................. 163
  $rev->type -> string .............................................................................................. 163
  $rev->user -> string .............................................................................................. 163
  Static Methods ..................................................................................................... 163
  Instance Methods ................................................................................................ 163

Class P4_Integration ............................................................................................... 164
  Description ............................................................................................................. 164
  Properties .............................................................................................................. 164
  $integ->how -> string ........................................................................................... 164
  $integ->file -> string ............................................................................................ 164
  $integ->srev -> int .................................................................................................. 164
  $integ->srev -> int .................................................................................................. 164
APIs for Scripting

Static Methods ................................................................. 164
Instance Methods ............................................................. 164
Class P4_Map ........................................................................ 165
Description ........................................................................ 165
Properties ........................................................................... 165
Constructor ........................................................................... 165
P4_Map::__construct( [ array ] ) -> P4_Map ............................ 165
Static Methods ...................................................................... 165
P4_Map::join ( map1, map2 ) -> P4_Map .................................. 165
Instance Methods ............................................................. 165
$map->clear() -> void ......................................................... 165
$map->count() -> int ......................................................... 165
$map->is_empty() -> bool ................................................... 166
$map->insert( string ... ) -> void ........................................ 166
$map->translate ( string, [ bool ] )-> string ......................... 166
$map->includes( string ) -> bool ......................................... 166
$map->reverse() -> P4_Map ................................................ 166
$map->lhs() -> array ........................................................ 166
$map->rhs() -> array ........................................................ 166
$map->as_array() -> array ................................................ 167
Class P4_MergeData ............................................................ 168
Description ........................................................................ 168
Properties ........................................................................... 168
$md->your_name -> string .................................................... 168
$md->their_name -> string .................................................... 168
$md->base_name -> string .................................................... 168
$md->your_path -> string ..................................................... 168
$md->their_path -> string ..................................................... 168
$md->base_path -> string ..................................................... 168
$md->result_path -> string .................................................... 168
$md->merge_hint -> string ..................................................... 168
Class P4_OutputHandlerAbstract .......................................... 169
Description ........................................................................ 169
Class Methods ..................................................................... 169
class MyHandler extends P4_OutputHandlerAbstract .................. 169
Instance Methods ............................................................. 169
$handler->outputBinary -> int ............................................. 169
$handler->outputInfo -> int ............................................... 169
$handler->outputMessage -> int ............................................ 169
$handler->outputStat -> int ............................................... 169
$handler->outputText -> int ............................................... 169
Class P4_Resolver .................................................................. 170
Description ........................................................................ 170
Properties ........................................................................... 170
Static Methods ..................................................................... 170
Instance Methods ............................................................. 170
$resolver->resolve( self, mergeData ) -> string ......................... 170
License Statements ........................................................................................................................................ 171
Chapter 1  About This Manual

This guide contains details about using the derived APIs for Ruby, Perl, Python, and PHP to create scripts that interact correctly with the Perforce versioning service. You can download these APIs from the Perforce web site:

http://www.perforce.com/product/components/apis

These derived APIs depend on the C/C++ API. See the Perforce C/C++ API User’s Guide for details.

Please give us feedback

If you have any feedback for us, or detect any errors in this guide, please email details to <manual@perforce.com>. 
Introduction

P4Ruby is an extension to the Ruby programming language that allows you to run Perforce commands from within Ruby scripts, and get the results in a Ruby-friendly format.

The main features are:

• Get Perforce data and forms in hashes and arrays.
• Edit Perforce forms by modifying hashes.
• Exception based error handling.
• Controllable handling of warnings such as "File(s) up-to-date." on a sync.
• Run as many commands on a connection as required.
• The output of a command is returned as a Ruby array. For non-tagged output, the elements of the array are strings. For tagged output, the elements of the array are Ruby hashes. For forms, the output is an array of `P4::Spec` objects.
• Thread-safe and thread-friendly; you can have multiple instances of the `P4` class running in different threads.
• Exception-based error handling. Trap `P4Exceptions` for complete, high-level error handling.

The most recent release of P4Ruby is 2014.1.

System Requirements

P4Ruby is supported on Windows, Linux, Solaris, OS X, and FreeBSD.

To build P4Ruby, your development machine must also have:

• Ruby 1.8 or 1.9 development files.
• `make` (or `nmake` on Windows).
• The 2014.1 Perforce C/C++ API for your target platform. Older releases might work, but are not supported.
• The same C++ compiler used to build the Perforce C++ API on your target platform.

(If you get "unresolved symbol" errors when building or running P4Ruby, you probably used the wrong compiler or the wrong Perforce API build.)

Installing P4Ruby

As of version 2015.1, the recommended mechanism for installing P4Ruby is via gems.

Outside of Windows, the `p4ruby` gem installs must be compiled locally against your installation of Ruby. If you can build the core Ruby distribution locally, you likely can install P4Ruby without incident. On Windows, precompiled gems will be made available.
$ gem install p4ruby -- --with-p4api-dir=DIR

In the example above, the \textit{DIR} is the path to a local copy of the Perforce C++ API distribution. The Perforce C++ API should match the major and minor version of P4Ruby. If you omit the \textit{--with-p4api-dir} option, the gem attempts to download a version of the API itself from ftp.perforce.com.

More installation options are described in the P4Ruby project in the Perforce Workshop:

\section*{Programming with P4Ruby}

The following example shows how to create a new client workspace based on an existing template:

\begin{verbatim}
require "P4"
template = "my-client-template"
client_root = 'c:\p4-work'
p4 = P4.new
p4.connect
begin
  # Run a "p4 client -t template -o" and convert it into a Ruby hash
  spec = p4.fetch_client( "-t", template, "my-new-client")
  # Now edit the fields in the form
  spec[ "Root" ] = client_root
  spec[ "Options" ] = spec[ "Options" ].sub( "normdir", "rmdir" )
  # Now save the updated spec
  p4.save_client( spec )
  # Point to the newly-created client
  p4.client="my-new-client"
  # And sync it.
  p4.run_sync
rescue P4Exception
  # If any errors occur, we'll jump in here. Just log them
  # and raise the exception up to the higher level
  p4.errors.each { |e| $stderr.puts( e ) } raise
end
\end{verbatim}

\section*{Connecting to SSL-enabled servers}

Scripts written with P4Ruby use any existing \texttt{P4TRUST} file present in their operating environment (by default, \texttt{.p4trust} in the home directory of the user that runs the script).
If the fingerprint returned by the server fails to match the one installed in the P4TRUST file associated with the script’s run-time environment, your script will (and should!) fail to connect to the server.

**P4Ruby classes**

The P4 module consists of several public classes:

- “P4” on page 5
- “P4Exception” on page 9
- “P4::DepotFile” on page 9
- “P4::Revision” on page 9
- “P4::Integration” on page 10
- “P4::Map” on page 10
- “P4::MergeData” on page 11
- “P4::Message” on page 11
- “P4::OutputHandler” on page 11
- “P4::Progress” on page 12
- “P4::Spec” on page 12

The following tables provide brief details about each public class.

### P4

The main class used for executing Perforce commands. Almost everything you do with P4Ruby will involve this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>identify</code></td>
<td>Return the version of P4Ruby in use (class method).</td>
</tr>
<tr>
<td><code>new</code></td>
<td>Construct a new P4 object (class method).</td>
</tr>
<tr>
<td><code>api_level=</code></td>
<td>Set desired API compatibility level.</td>
</tr>
<tr>
<td><code>api_level</code></td>
<td>Return current API compatibility level.</td>
</tr>
<tr>
<td><code>at_exception_level</code></td>
<td>Execute the associated block under a specific exception level, returning to previous exception level when block returns.</td>
</tr>
<tr>
<td><code>charset=</code></td>
<td>Set character set when connecting to Unicode servers.</td>
</tr>
<tr>
<td><code>charset</code></td>
<td>Get character set when connecting to Unicode servers.</td>
</tr>
<tr>
<td><code>client=</code></td>
<td>Set client workspace (P4CLIENT).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>client</code></td>
<td>Get current client workspace (<code>P4CLIENT</code>).</td>
</tr>
<tr>
<td><code>connect</code></td>
<td>Connect to the Perforce Server, raise <code>P4Exception</code> on failure.</td>
</tr>
<tr>
<td><code>connected?</code></td>
<td>Test whether or not session has been connected and/or has been dropped.</td>
</tr>
<tr>
<td><code>cwd=</code></td>
<td>Set current working directory.</td>
</tr>
<tr>
<td><code>cwd</code></td>
<td>Get current working directory.</td>
</tr>
<tr>
<td><code>delete &lt;spectype&gt;</code></td>
<td>Shortcut methods for deleting clients, labels, etc.</td>
</tr>
<tr>
<td><code>disconnect</code></td>
<td>Disconnect from the Perforce Server.</td>
</tr>
<tr>
<td><code>each &lt;spectype&gt;</code></td>
<td>Shortcut methods for iterating through clients, labels, etc.</td>
</tr>
<tr>
<td><code>env</code></td>
<td>Get the value of a Perforce environment variable, taking into account <code>P4CONFIG</code> files and (on Windows or OS X) the registry or user preferences.</td>
</tr>
<tr>
<td><code>errors</code></td>
<td>Return the array of errors that occurred during execution of previous command.</td>
</tr>
<tr>
<td><code>exception_level=</code></td>
<td>Control which types of events give rise to exceptions (<code>P4::RAISE_NONE</code>, <code>RAISE_ERRORS</code>, or <code>RAISE_ALL</code>).</td>
</tr>
<tr>
<td><code>exception_level</code></td>
<td>Return the current exception level.</td>
</tr>
<tr>
<td><code>fetch &lt;spectype&gt;</code></td>
<td>Shortcut methods for retrieving the definitions of clients, labels, etc.</td>
</tr>
<tr>
<td><code>format_spec</code></td>
<td>Convert fields in a hash containing the elements of a Perforce form (spec) into the string representation familiar to users.</td>
</tr>
<tr>
<td><code>format &lt;spectype&gt;</code></td>
<td>Shortcut method; equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>p4.format_spec( &quot;&lt;spectype&gt;&quot;, aHash )</code></td>
</tr>
<tr>
<td><code>handler=</code></td>
<td>Set output handler.</td>
</tr>
<tr>
<td><code>handler</code></td>
<td>Get output handler.</td>
</tr>
<tr>
<td><code>host=</code></td>
<td>Set the name of the current host (<code>P4HOST</code>).</td>
</tr>
<tr>
<td><code>host</code></td>
<td>Get the current hostname.</td>
</tr>
<tr>
<td><code>input=</code></td>
<td>Store input for next command.</td>
</tr>
<tr>
<td><code>maxlocktime=</code></td>
<td>Set MaxLockTime used for all following commands.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>maxlocktime</code></td>
<td>Get MaxLockTime used for all following commands.</td>
</tr>
<tr>
<td><code>maxresults=</code></td>
<td>Set MaxResults used for all following commands.</td>
</tr>
<tr>
<td><code>maxresults</code></td>
<td>Get MaxResults used for all following commands.</td>
</tr>
<tr>
<td><code>maxscanrows=</code></td>
<td>Set MaxScanRows used for all following commands.</td>
</tr>
<tr>
<td><code>maxscanrows</code></td>
<td>Get MaxScanRows used for all following commands.</td>
</tr>
<tr>
<td><code>messages</code></td>
<td>Returns all messages from the server as <code>P4::Message</code> objects.</td>
</tr>
<tr>
<td><code>p4config_file</code></td>
<td>Get the location of the configuration file used (<code>P4CONFIG</code>).</td>
</tr>
<tr>
<td><code>parse_&lt;spectype&gt;</code></td>
<td>Shortcut method; equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>p4.parse_spec(&quot;&lt;spectype&gt;&quot;, aString)</code></td>
</tr>
<tr>
<td><code>parse_spec</code></td>
<td>Parses a Perforce form (spec) in text form into a Ruby hash using the spec</td>
</tr>
<tr>
<td></td>
<td>definition obtained from the server.</td>
</tr>
<tr>
<td><code>password=</code></td>
<td>Set Perforce password (<code>P4PASSWD</code>).</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Get the current password or ticket.</td>
</tr>
<tr>
<td><code>port=</code></td>
<td>Set host and port (<code>P4PORT</code>).</td>
</tr>
<tr>
<td><code>port</code></td>
<td>Get host and port (<code>P4PORT</code>) of the current Perforce server.</td>
</tr>
<tr>
<td><code>prog=</code></td>
<td>Set program name as shown by <code>p4 monitor show -e</code>.</td>
</tr>
<tr>
<td><code>prog</code></td>
<td>Get program name as shown by <code>p4 monitor show -e</code>.</td>
</tr>
<tr>
<td><code>progress=</code></td>
<td>Set progress indicator.</td>
</tr>
<tr>
<td><code>progress</code></td>
<td>Get progress indicator.</td>
</tr>
<tr>
<td><code>run_cmd</code></td>
<td>Shortcut method; equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>p4.run(&quot;cmd&quot;, arguments...)</code></td>
</tr>
<tr>
<td><code>run</code></td>
<td>Runs the specified Perforce command with the arguments supplied.</td>
</tr>
<tr>
<td><code>run_filelog</code></td>
<td>Runs a <code>p4 filelog</code> on the fileSpec provided, returns an array of <code>P4::DepotFile</code> objects.</td>
</tr>
<tr>
<td><code>run_login</code></td>
<td>Runs <code>p4 login</code> using a password or ticket set by the user.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run_password</td>
<td>A thin wrapper to make it easy to change your password.</td>
</tr>
<tr>
<td>run_resolve</td>
<td>Interface to <code>p4 resolve</code>.</td>
</tr>
<tr>
<td>run_submit</td>
<td>Submit a changelist to the server.</td>
</tr>
<tr>
<td>run_tickets</td>
<td>Get a list of tickets from the local tickets file.</td>
</tr>
<tr>
<td>save <code>&lt;spectype&gt;</code></td>
<td>Shortcut method; equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>p4.input = hashOrString</code></td>
</tr>
<tr>
<td></td>
<td><code>p4.run( &quot;&lt;spectype&gt;&quot;, &quot;-i&quot; )</code></td>
</tr>
<tr>
<td>server_case_sensitive?</td>
<td>Detects whether or not the server is case sensitive.</td>
</tr>
<tr>
<td>server_level</td>
<td>Returns the current Perforce server level.</td>
</tr>
<tr>
<td>server_unicode?</td>
<td>Detects whether or not the server is in unicode mode.</td>
</tr>
<tr>
<td>set_env</td>
<td>On Windows or OS X, set a variable in the registry or user preferences.</td>
</tr>
<tr>
<td>streams=</td>
<td>Enable or disable support for streams.</td>
</tr>
<tr>
<td>streams?</td>
<td>Test whether or not the server supports streams</td>
</tr>
<tr>
<td>tagged</td>
<td>Toggles tagged output (true or false). By default, tagged output is on.</td>
</tr>
<tr>
<td>tagged=</td>
<td>Sets tagged output. By default, tagged output is on.</td>
</tr>
<tr>
<td>tagged?</td>
<td>Detects whether or not tagged output is enabled.</td>
</tr>
<tr>
<td>ticketfile=</td>
<td>Set the location of the <code>P4TICKETS</code> file.</td>
</tr>
<tr>
<td>ticketfile</td>
<td>Get the location of the <code>P4TICKETS</code> file.</td>
</tr>
<tr>
<td>track=</td>
<td>Activate or disable server performance tracking.</td>
</tr>
<tr>
<td>track?</td>
<td>Detect whether server performance tracking is active.</td>
</tr>
<tr>
<td>track_output</td>
<td>Returns server tracking output.</td>
</tr>
<tr>
<td>user=</td>
<td>Set the Perforce username (<code>P4USER</code>).</td>
</tr>
<tr>
<td>user</td>
<td>Get the Perforce username (<code>P4USER</code>).</td>
</tr>
<tr>
<td>version=</td>
<td>Set your script’s version as reported to the server.</td>
</tr>
<tr>
<td>version</td>
<td>Get your script’s version as reported by the server.</td>
</tr>
</tbody>
</table>
## Chapter 2. P4Ruby

### APIs for Scripting

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>warnings</td>
<td>Returns the array of warnings that arose during execution of the last command.</td>
</tr>
</tbody>
</table>

**P4Exception**

Used as part of error reporting and is derived from the Ruby `RuntimeError` class.

**P4::DepotFile**

Utility class allowing access to the attributes of a file in the depot. Returned by `P4#run_filelog()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>depot_file</td>
<td>Name of the depot file to which this object refers.</td>
</tr>
<tr>
<td>each_revision</td>
<td>Iterates over each revision of the depot file.</td>
</tr>
<tr>
<td>revisions</td>
<td>Returns an array of revision objects for the depot file.</td>
</tr>
</tbody>
</table>

**P4::Revision**

Utility class allowing access to the attributes of a revision `P4::DepotFile` object. Returned by `P4#run_filelog()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Action that created the revision.</td>
</tr>
<tr>
<td>change</td>
<td>Changelist number.</td>
</tr>
<tr>
<td>client</td>
<td>Client workspace used to create this revision.</td>
</tr>
<tr>
<td>depot_file</td>
<td>Name of the file in the depot.</td>
</tr>
<tr>
<td>desc</td>
<td>Short changelist description.</td>
</tr>
<tr>
<td>digest</td>
<td>MD5 digest of this revision.</td>
</tr>
<tr>
<td>filesize</td>
<td>Returns the size of this revision.</td>
</tr>
<tr>
<td>integrations</td>
<td>Array of <code>P4::Integration</code> objects.</td>
</tr>
<tr>
<td>rev</td>
<td>Revision number.</td>
</tr>
<tr>
<td>time</td>
<td>Timestamp.</td>
</tr>
<tr>
<td>type</td>
<td>Perforce file type.</td>
</tr>
</tbody>
</table>
## Chapter 2. P4Ruby

### Method | Description
--- | ---
user | User that created this revision.

### P4::Integration

Utility class allowing access to the attributes of an integration record for a `P4::Revision` object. Returned by `P4#run_filelog()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>Integration method (merge/branch/copy/ignored).</td>
</tr>
<tr>
<td>file</td>
<td>Integrated file.</td>
</tr>
<tr>
<td>srev</td>
<td>Start revision.</td>
</tr>
<tr>
<td>erev</td>
<td>End revision.</td>
</tr>
</tbody>
</table>

### P4::Map

A class that allows users to create and work with Perforce mappings without requiring a connection to the Perforce Server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new</td>
<td>Construct a new map object (class method).</td>
</tr>
<tr>
<td>join</td>
<td>Joins two maps to create a third (class method).</td>
</tr>
<tr>
<td>clear</td>
<td>Empties a map.</td>
</tr>
<tr>
<td>count</td>
<td>Returns the number of entries in a map.</td>
</tr>
<tr>
<td>empty?</td>
<td>Tests whether or not a map object is empty.</td>
</tr>
<tr>
<td>insert</td>
<td>Inserts an entry into the map.</td>
</tr>
<tr>
<td>translate</td>
<td>Translate a string through a map.</td>
</tr>
<tr>
<td>includes?</td>
<td>Tests whether a path is mapped.</td>
</tr>
<tr>
<td>reverse</td>
<td>Returns a new mapping with the left and right sides reversed.</td>
</tr>
<tr>
<td>lhs</td>
<td>Returns the left side as an array.</td>
</tr>
<tr>
<td>rhs</td>
<td>Returns the right side as an array.</td>
</tr>
<tr>
<td>to_a</td>
<td>Returns the map as an array.</td>
</tr>
</tbody>
</table>
P4::MergeData

Class encapsulating the context of an individual merge during execution of a `p4 resolve` command. Passed as a parameter to the block passed to `P4#run_resolve()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>your_name</td>
<td>Returns the name of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td>their_name</td>
<td>Returns the name of &quot;their&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td>base_name</td>
<td>Returns the name of &quot;base&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td>your_path</td>
<td>Returns the path of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td>their_path</td>
<td>Returns the path of &quot;their&quot; file in the merge. (temporary file on workstation into which <code>their_name</code> has been loaded)</td>
</tr>
<tr>
<td>base_path</td>
<td>Returns the path of the base file in the merge. (temporary file on workstation into which <code>base_name</code> has been loaded)</td>
</tr>
<tr>
<td>result_path</td>
<td>Returns the path to the merge result. (temporary file on workstation into which the automatic merge performed by the server has been loaded)</td>
</tr>
<tr>
<td>merge_hint</td>
<td>Returns hint from server as to how user might best resolve merge.</td>
</tr>
<tr>
<td>run_merge</td>
<td>If the environment variable <code>P4MERGE</code> is defined, run it and return a boolean based on the return value of that program.</td>
</tr>
</tbody>
</table>

P4::Message

Utility class allowing access to the attributes of a message object returned by `P4#messages()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>severity</td>
<td>Returns the severity of the message.</td>
</tr>
<tr>
<td>generic</td>
<td>Returns the generic class of the error.</td>
</tr>
<tr>
<td>msgid</td>
<td>Returns the unique ID of the error message.</td>
</tr>
<tr>
<td>to_s</td>
<td>Returns the error message as a string.</td>
</tr>
<tr>
<td>inspect</td>
<td>Converts the error object into a string for debugging purposes.</td>
</tr>
</tbody>
</table>

P4::OutputHandler

Handler class that provides access to streaming output from the server; set `P4#handler()` to an instance of a subclass of `P4::OutputHandler` to enable callbacks:
## Chapter 2. P4Ruby

### APIs for Scripting

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outputBinary</code></td>
<td>Process binary data</td>
</tr>
<tr>
<td><code>outputInfo</code></td>
<td>Process tabular data</td>
</tr>
<tr>
<td><code>outputMessage</code></td>
<td>Process information or errors.</td>
</tr>
<tr>
<td><code>outputStat</code></td>
<td>Process tagged output.</td>
</tr>
<tr>
<td><code>outputText</code></td>
<td>Process text data</td>
</tr>
</tbody>
</table>

### P4::Progress

Handler class that provides access to progress indicators from the server; set `P4#progress()` to an instance of a subclass of `P4::Progress` with the following methods (even if the implementations are empty) to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init</code></td>
<td>Initialize progress indicator as designated type.</td>
</tr>
<tr>
<td><code>total</code></td>
<td>Total number of units (if known).</td>
</tr>
<tr>
<td><code>description</code></td>
<td>Description and type of units to be used for progress reporting.</td>
</tr>
<tr>
<td><code>update</code></td>
<td>If non-zero, user has requested a cancellation of the operation.</td>
</tr>
<tr>
<td><code>done</code></td>
<td>If non-zero, operation has failed.</td>
</tr>
</tbody>
</table>

### P4::Spec

Subclass of hash allowing access to the fields in a Perforce specification form. Also checks that the fields that are set are valid fields for the given type of spec. Returned by `P4#fetch_<spectype>()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spec.fieldname</code></td>
<td>Return the value associated with the field named <code>fieldname</code>.</td>
</tr>
<tr>
<td><code>spec.fieldname=</code></td>
<td>Set the value associated with the field named <code>fieldname</code>.</td>
</tr>
<tr>
<td><code>spec.permitted_fields</code></td>
<td>Returns an array containing the names of fields that are valid in this spec object.</td>
</tr>
</tbody>
</table>
Class P4

Description

Main interface to the Perforce client API. Each P4 object provides you with a thread-safe API level interface to Perforce. The basic model is to:

1. Instantiate your P4 object.
2. Specify your Perforce client environment.
   - client
   - host
   - password
   - port
   - user
3. Set any options to control output or error handling:
   - exception_level
4. Connect to the Perforce service.
   
   The Perforce protocol is not designed to support multiple concurrent queries over the same connection. Multithreaded applications that use the C++ API or derived APIs (including P4Ruby) should ensure that a separate connection is used for each thread, or that only one thread may use a shared connection at a time.
5. Run your Perforce commands.
6. Disconnect from the Perforce service.

Class Methods

P4.identify -> aString

Return the version of P4Ruby that you are using. Also reports the version of the OpenSSL library used for building the underlying Perforce C++ API with which P4Ruby was built.

```ruby
ruby -rP4 -e 'puts( P4.identify )'
```

Some of this information is already made available through the predefined constants P4::VERSION, P4::OS, and P4::PATCHLEVEL.

P4.new -> aP4

Constructs a new P4 object.
p4 = P4.new()

### Instance Methods

**p4.api_level= anInteger -> anInteger**

Sets the API compatibility level desired. This is useful when writing scripts using Perforce commands that do not yet support tagged output. In these cases, upgrading to a later server that supports tagged output for the commands in question can break your script. Using this method allows you to lock your script to the output format of an older Perforce release and facilitate seamless upgrades. This method *must* be called prior to calling `P4#connect()`.

```ruby
p4 = P4.new
p4.api_level = 67 # Lock to 2010.1 format
p4.connect
... 
```

For the API integer levels that correspond to each Perforce release, see:

[http://kb.perforce.com/article/512](http://kb.perforce.com/article/512)

**p4.api_level -> anInteger**

Returns the current Perforce API compatibility level. Each iteration of the Perforce Server is given a level number. As part of the initial communication, the client protocol level is passed between client application and the Perforce Server. This value, defined in the Perforce API, determines the communication protocol level that the Perforce client will understand. All subsequent responses from the Perforce Server can be tailored to meet the requirements of that client protocol level.

For more information, see:

[http://kb.perforce.com/article/512](http://kb.perforce.com/article/512)

**p4.at_exception_level( lev ) { ... } -> self**

Executes the associated block under a specific exception level. Returns to the previous exception level when the block returns.

```ruby
p4 = P4.new
p4.client = "www"
p4.connect

p4.at_exception_level(P4::RAISE_ERRORS) do
  p4.run_sync
end

p4.disconnect
```


**p4.charset** = **aString -> aString**

Sets the character set to use when connect to a Unicode enabled server. Do not use when working with non-Unicode-enabled servers. By default, the character set is the value of the `P4CHARSET` environment variable. If the character set is invalid, this method raises a `P4Exception`.

```ruby
p4 = P4.new
p4.client = "www"
p4.charset = "iso8859-1"
p4.connect
p4.run_sync
p4.disconnect
```

**p4.charset -> aString**

Get the name of the character set in use when working with Unicode-enabled servers.

```ruby
p4 = P4.new
p4.charset = "utf8"
puts( p4.charset )
```

**p4.client** = **aString -> aString**

Set the name of the client workspace you wish to use. If not called, defaults to the value of `P4CLIENT` taken from any `P4CONFIG` file present, or from the environment as per the usual Perforce convention. Must be called before connecting to the Perforce server.

```ruby
p4 = P4.new
p4.client = "www"
p4.connect
p4.run_sync
p4.disconnect
```

**p4.client -> aString**

Get the name of the Perforce client currently in use.

```ruby
p4 = P4.new
puts( p4.client )
```

**p4.connect -> aBool**

Connect to the Perforce Server. You must connect before you can execute commands. Raises a `P4Exception` if the connection attempt fails.

```ruby
p4 = P4.new
```
p4 = P4.new
p4.connect

**p4.connected? -> aBool**

Test whether or not the session has been connected, and if the connection has not been dropped.

```ruby
p4 = P4.new
p4.connected?
```

**p4.cwd = aString -> aString**

Sets the current working directly. Can be called prior to executing any Perforce command. Sometimes necessary if your script executes a `chdir()` as part of its processing.

```ruby
p4 = P4.new
p4.cwd = "/home/bruno"
```

**p4.cwd -> aString**

Get the current working directory.

```ruby
p4 = P4.new
puts( p4.cwd )
```

**p4.delete_<spectype>( [options], name ) -> anArray**

The delete methods are simply shortcut methods that allow you to quickly delete the definitions of clients, labels, branches, etc. These methods are equivalent to:

```ruby
p4.run( "<spectype>", '-d', [options], "spec name" )
```

For example:
Chapter 2. P4Ruby

```ruby
require "P4"
require "parsedate"
include ParseDate
now = Time.now
p4 = P4.new
begin
  p4.connect
  p4.run_clients.each do |client|
    atime = parsedate(client[ "Access" ])
    if (atime + 24 * 3600 * 365) < now
      p4.delete_client( '-f', client[ "client" ])
    end
  end
rescue P4Exception
  p4.errors.each { |e| puts(e) }
ensure
  p4.disconnect
end
```

**p4.disconnect -> true**

Disconnect from the Perforce Server.

```ruby
p4 = P4.new
p4.connect
p4.disconnect
```

**p4.each_<spectype>( arguments ) -> anArray**

The `p4.each_<spectype>` methods are shortcut methods that allow you to quickly iterate through clients, labels, branches, etc. Valid `<spectype>`s are `clients, labels, branches, changes, streams, jobs, users, groups, depots` and `servers`. Valid arguments are any arguments that would be valid for the corresponding `run_<spectype>` command.

For example, to iterate through clients:

```ruby
p4.each_clients do |c|
  # work with the retrieved client spec
end
```

is equivalent to:

```ruby
```
clients = p4.run_clients
clients.each do |c|
  client = p4.fetch_client( c['client'] )
  # work with the retrieved client spec
end

# API for Scripting

**p4.env -> string**

Get the value of a Perforce environment variable, taking into account `P4CONFIG` files and (on Windows and OS X) the registry or user preferences.

```ruby
p4 = P4.new
puts p4.env( "P4PORT" )
```

**p4.errors -> anArray**

Returns the array of errors which occurred during execution of the previous command.

```ruby
p4 = P4.new
begin
  p4.connect
  p4.exception_level( P4::RAISE_ERRORS ) # ignore "File(s) up-to-date"
  files = p4.run_sync
rescue P4Exception
  p4.errors.each { |e| puts( e ) }
ensure
  p4.disconnect
end
```

**p4.exception_level= anInteger -> anInteger**

Configures the events which give rise to exceptions. The following three levels are supported:

- **P4::RAISE_NONE** disables all exception raising and makes the interface completely procedural.
- **P4::RAISE_ERRORS** causes exceptions to be raised only when errors are encountered.
- **P4::RAISE_ALL** causes exceptions to be raised for both errors and warnings. This is the default.

```ruby
p4 = P4.new
p4.exception_level = P4::RAISE_ERRORS
p4.connect   # P4Exception on failure
p4.run_sync  # File(s) up-to-date is a warning so no exception is raised
p4.disconnect
```

**p4.exception_level -> aNumber**

Returns the current exception level.
**p4.fetch_<spectype>( [name] ) -> aP4::Spec**

The `fetch_<spectype>` methods are shortcut methods that allow you to quickly fetch the definitions of clients, labels, branches, etc. They’re equivalent to:

```
p4.run( "<spectype>", '-o', ... ).shift
```

For example:

```ruby
p4 = P4.new
begin
  p4.connect
  client       = p4.fetch_client()
  other_client = p4.fetch_client( "other" )
  label        = p4.fetch_label( "somelabel" )
rescue P4Exception
  p4.errors.each { |e| puts( e ) }
ensure
  p4.disconnect
end
```

**p4.format_spec( "<spectype>", aHash ) -> aString**

Converts the fields in a hash containing the elements of a Perforce form (spec) into the string representation familiar to users.

The first argument is the type of spec to format: for example, `client`, `branch`, `label`, and so on. The second argument is the hash to parse.

There are shortcuts available for this method. You can use:

```
p4.format_<spectype>( hash )
```

instead of:

```
p4.format_spec( "<spectype>", hash )
```

where `<spectype>` is the name of a Perforce spec, such as `client`, `label`, etc.

**p4.format_<spectype> aHash -> aHash**

The `format_<spectype>` methods are shortcut methods that allow you to quickly fetch the definitions of clients, labels, branches, etc. They’re equivalent to:

```
p4.format_spec( "<spectype>", aHash )
```
p4.handler= aHandler -> aHandler

Set the current output handler. This should be a subclass of P4::OutputHandler.

p4.handler -> aHandler

Get the current output handler.

p4.host= aString -> aString

Set the name of the current host. If not called, defaults to the value of P4HOST taken from any P4CONFIG file present, or from the environment as per the usual Perforce convention. Must be called before connecting to the Perforce server.

```ruby
p4 = P4.new
p4.host = "workstation123.perforce.com"
p4.connect
...  
p4.disconnect
```

p4.host -> aString

Get the current hostname.

```ruby
p4 = P4.new
puts( p4.host )
```

p4.input= ( aString|aHash|anArray ) -> aString|aHash|anArray

Store input for the next command.

Call this method prior to running a command requiring input from the user. When the command requests input, the specified data will be supplied to the command. Typically, commands of the form `p4 cmd -i` are invoked using the `P4#save_<spectype>()` methods, which call `P4#input()` internally; there is no need to call `P4#input()` when using the `P4#save_<spectype>()` shortcuts.

You may pass a string, a hash, or (for commands that take multiple inputs from the user) an array of strings or hashes. If you pass an array, note that the array will be shifted each time Perforce asks the user for input.
p4 = P4.new
p4.connect

change = p4.run_change("-o").shift
change["Description"] = "Autosubmitted changelist"

p4.input = change
p4.run_submit("-i")
p4.disconnect

p4.maxlocktime = anInteger -> anInteger

Limit the amount of time (in milliseconds) spent during data scans to prevent the server from locking
tables for too long. Commands that take longer than the limit will be aborted. The limit remains
in force until you disable it by setting it to zero. See `p4 help maxlocktime` for information on the
commands that support this limit.

p4 = P4.new
begin
  p4.connect
  p4.maxlocktime = 10000 # 10 seconds
  files = p4.run_sync
rescue P4Exception => ex
  p4.errors.each { |e| $stderr.puts(e) }
ensure
  p4.disconnect
end

p4.maxlocktime -> anInteger

Get the current maxlocktime setting.

p4 = P4.new
puts(p4.maxlocktime)

p4.maxresults = anInteger -> anInteger

Limit the number of results Perforce permits for subsequent commands. Commands that produce
more than this number of results will be aborted. The limit remains in force until you disable it by
setting it to zero. See `p4 help maxresults` for information on the commands that support this limit.
Chapter 2. P4Ruby

p4 = P4.new
begin
  p4.connect
  p4.maxresults = 100
  files = p4.run_sync
rescue P4Exception => ex
  p4.errors.each { |e| $stderr.puts( e ) }
ensure
  p4.disconnect
end

p4.maxresults -> anInteger

Get the current maxresults setting.

p4 = P4.new
puts( p4.maxresults )

p4.maxscanrows= anInteger -> anInteger

Limit the number of database records Perforce will scan for subsequent commands. Commands that attempt to scan more than this number of records will be aborted. The limit remains in force until you disable it by setting it to zero. See p4 help maxscanrows for information on the commands that support this limit.

p4 = P4.new
begin
  p4.connect
  p4.maxscanrows = 100
  files = p4.run_sync
rescue P4Exception => ex
  p4.errors.each { |e| $stderr.puts( e ) }
ensure
  p4.disconnect
end

p4.maxscanrows -> anInteger

Get the current maxscanrows setting.

p4 = P4.new
puts( p4.maxscanrows )

p4.messages -> aP4::Message

Returns a message from the Perforce Server in the form of a P4::Message object.
Chapter 2. P4Ruby

```ruby
p4 = P4.new
p4.exception_level = P4::RAISE_NONE
p4.run_sync
p4.run_sync # this second sync should return "File(s) up-to-date."
w = p4.messages[0]
puts ( w.to_s )
```

### p4.p4config_file -> aString

Get the path to the current P4CONFIG file.

```ruby
p4 = P4.new
puts( p4.p4config_file )
```

### p4.parse_<spectype>( aString ) -> aP4::Spec

This is equivalent to:

```ruby
p4.parse_spec( "<spectype>", aString )
```

### p4.parse_spec( "<spectype>", aString ) -> aP4::Spec

Parses a Perforce form (spec) in text form into a Ruby hash using the spec definition obtained from the server.

The first argument is the type of spec to parse: client, branch, label, and so on. The second argument is the string buffer to parse.

Note that there are shortcuts available for this method. You can use:

```ruby
p4.parse_<spectype>( buf )
```

instead of:

```ruby
p4.parse_spec( "<spectype>", buf )
```

Where `<spectype>` is one of client, branch, label, and so on.

### p4.password= aString -> aString

Set your Perforce password, in plain text. If not used, takes the value of P4PASSWD from any P4CONFIG file in effect, or from the environment according to the normal Perforce conventions. This password will also be used if you later call `p4.run_login` to login using the 2003.2 and later ticket system.
p4 = P4.new
p4.password = "mypass"
p4.connect
p4.run_login

p4.password -> aString

Get the current password or ticket. This may be the password in plain text, or if you've used P4#run_login(), it'll be the value of the ticket you've been allocated by the server.

p4 = P4.new
puts( p4.password )

p4.port = aString -> aString

Set the host and port of the Perforce server you want to connect to. If not called, defaults to the value of P4PORT in any P4CONFIG file in effect, and then to the value of P4PORT taken from the environment.

p4 = P4.new
p4.port = "localhost:1666"
p4.connect
...
p4.disconnect

p4.port -> aString

Get the host and port of the current Perforce server.

p4 = P4.new
puts( p4.port )

p4.prog = aString -> aString

Set the name of the program, as reported to Perforce system administrators running p4 monitor show -e in Perforce 2004.2 or later releases.

p4 = P4.new
p4.prog = "sync-script"
p4.connect
...
p4.disconnect

p4.prog -> aString

Get the name of the program as reported to the Perforce Server.
Chapter 2. P4Ruby

```ruby
p4 = P4.new
p4.prog = "sync-script"
puts( p4.prog )
```

### p4.progress= aProgress -> aProgress

Set the current progress indicator. This should be a subclass of `P4::Progress`.

### p4.progress -> aProgress

Get the current progress indicator.

### p4.run_<cmd>:( arguments )-> anArray

This is equivalent to:

```ruby
p4.run( "cmd", arguments... )
```

### p4.run( aCommand, arguments... ) -> anArray

Base interface to all the run methods in this API. Runs the specified Perforce command with the arguments supplied. Arguments may be in any form as long as they can be converted to strings by `to_s`.

The `P4#run()` method returns an array of results whether the command succeeds or fails; the array may, however, be empty. Whether the elements of the array are strings or hashes depends on (a) server support for tagged output for the command, and (b) whether tagged output was disabled by calling `p4.tagged = false`.

In the event of errors or warnings, and depending on the exception level in force at the time, `P4#run()` will raise a `P4Exception`. If the current exception level is below the threshold for the error/warning, `P4#run()` returns the output as normal and the caller must explicitly review `P4#errors()` and `P4#warnings()` to check for errors or warnings.

```ruby
p4 = P4.new
p4.connect
spec = p4.run( "client", "-o" ).shift
p4.disconnect
```

Shortcuts are available for `P4#run()`. For example:

```ruby
p4.run_command( args )
```

is equivalent to:
p4.run( "command", args )

There are also some shortcuts for common commands such as editing Perforce forms and submitting. Consequently, this:

```ruby
p4 = P4.new
p4.connect
clientspec = p4.run_client( "-o" ).shift
clientspec[ "Description" ] = "Build client"
p4.input = clientspec
p4.run_client( "-i" )
p4.disconnect
```

may be shortened to:

```ruby
p4 = P4.new
p4.connect
clientspec = p4.fetch_client
clientspec[ "Description" ] = "Build client"
p4.save_client( clientspec )
p4.disconnect
```

The following are equivalent:

```ruby
p4.delete_<spectype>() p4.run( "<spectype>", "-d" )
p4.fetch_<spectype>() p4.run( "<spectype>", "-o" ).shift
p4.save_<spectype>( spec ) p4.input = spec p4.run( "<spectype>", "-i" )
```

As the commands associated with P4#fetch_<spectype>() typically return only one item, these methods do not return an array, but instead return the first result element.

For convenience in submitting changelists, changes returned by P4#fetch_change() can be passed to P4#run_submit. For example:

```ruby
p4 = P4.new
p4.connect
spec = p4.fetch_changespec[ "Description" ] = "Automated change"
p4.run_submit( spec )
p4.disconnect
```

**p4.run_filelog( fileSpec ) -> anArray**

Runs a p4 filelog on the fileSpec provided and returns an array of P4::DepotFile results when executed in tagged mode, and an array of strings when executed in non-tagged mode. By default, the raw output of p4 filelog is tagged; this method restructures the output into a more user-friendly (and object-oriented) form.
p4 = P4.new
begin
    p4.connect
    p4.run_filelog( "index.html" ).shift.each_revision do |r|
        r.each_integration do |i|
            # Do something
        end
    end
rescue P4Exception
    p4.errors.each { |e| puts( e ) }
ensure
    p4.disconnect
end

p4.run_login( arg... ) -> anArray

Runs p4 login using a password or ticket set by the user.

p4.run_password( oldpass, newpass ) -> anArray

A thin wrapper to make it easy to change your password. This method is (literally) equivalent to the following code:

```ruby
p4.input( [ oldpass, newpass, newpass ] )
p4.run( "password" )
```

For example:

```ruby
p4 = P4.new
p4.password = "myoldpass"
begin
    p4.connect
    p4.run_password( "myoldpass", "mynewpass" )
rescue P4Exception
    p4.errors.each { |e| puts( e ) }
ensure
    p4.disconnect
end

p4.run_resolve( args ) [ block ] -> anArray

Interface to p4 resolve. Without a block, simply runs a non-interactive resolve (typically an automatic resolve).

```ruby
p4.run_resolve( "-at" )
```
When a block is supplied, the block is invoked once for each merge scheduled by Perforce. For each merge, a `P4::MergeData` object is passed to the block. This object contains the context of the merge.

The block determines the outcome of the merge by evaluating to one of the following strings:

<table>
<thead>
<tr>
<th>Block string</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ay</td>
<td>Accept Yours.</td>
</tr>
<tr>
<td>at</td>
<td>Accept Theirs.</td>
</tr>
<tr>
<td>am</td>
<td>Accept Merge result.</td>
</tr>
<tr>
<td>ae</td>
<td>Accept Edited result.</td>
</tr>
<tr>
<td>s</td>
<td>Skip this merge.</td>
</tr>
<tr>
<td>q</td>
<td>Abort the merge.</td>
</tr>
</tbody>
</table>

For example:

```ruby
p4.run_resolve() do
  |md|
  puts( "Merging..." )
  puts( "Yours: #{md.your_name}" )
  puts( "Theirs: #{md.their_name}" )
  puts( "Base: #{md.base_name}" )
  puts( "Yours file: #{md.your_path}" )
  puts( "Theirs file: #{md.their_path}" )
  puts( "Base file: #{md.base_path}" )
  puts( "Result file: #{md.result_path}" )
  puts( "Merge Hint: #{md.merge_hint}" )
  result = md.merge_hint
  if( result == "e" )
    puts( "Invoking external merge application" )
    result = "s"  # If the merge doesn't work, we'll skip
    result = "am" if md.run_merge()
  end
  result
end
```

### `p4.run_submit([aHash], [arg...]) -> anArray`

Submit a changelist to the server. To submit a changelist, set the fields of the changelist as required and supply any flags:

```ruby
change = p4.fetch_change
change_.description = "Some description"
p4.run_submit( "-r", change )
```
You can also submit a changelist by supplying the arguments as you would on the command line:

```
p4.run_submit( "-d", "Some description", "somedir/..." )
```

**p4.run_tickets() -> anArray**

Get a list of tickets from the local tickets file. Each ticket is a hash object with fields for **Host**, **User**, and **Ticket**.

**p4.save_<spectype>( hashOrString, [options] ) -> anArray**

The `save_<spectype>` methods are shortcut methods that allow you to quickly update the definitions of clients, labels, branches, etc. They are equivalent to:

```
p4.input = hashOrStringp4.run( "<spectype>", "-i" )
```

For example:

```ruby
p4 = P4.new
begin
  p4.connect
  client           = p4.fetch_client()
  client[ "Owner" ] = p4.user
  p4.save_client( client )
rescue P4Exception
  p4.errors.each { |e| puts( e ) }
ensure
  p4.disconnect
end
```

**p4.server_case_sensitive? -> aBool**

Detects whether or not the server is case-sensitive.

**p4.server_level -> anInteger**

Returns the current Perforce server level. Each iteration of the Perforce Server is given a level number. As part of the initial communication this value is passed between the client application and the Perforce Server. This value is used to determine the communication that the Perforce Server will understand. All subsequent requests can therefore be tailored to meet the requirements of this Server level.

For more information, see:

[http://kb.perforce.com/article/571](http://kb.perforce.com/article/571)

**p4.server_unicode? -> aBool**

Detects whether or not the server is in unicode mode.
p4.set_env= ( aString, aString ) -> aBool

On Windows or OS X, set a variable in the registry or user preferences. To unset a variable, pass an empty string as the second argument. On other platforms, an exception is raised.

```ruby
p4 = P4.new
p4.set_env = ( "P4CLIENT", "my_workspace" )
p4.set_env = ( "P4CLIENT", "" )
```

p4.streams= -> aBool

Enable or disable support for streams. By default, streams support is enabled at 2011.1 or higher (P4#api_level() >= 70). Raises a P4Exception if you attempt to enable streams on a pre-2011.1 server. You can enable or disable support for streams both before and after connecting to the server.

```ruby
p4 = P4.new
p4.streams = false
```

p4.streams? -> aBool

Detects whether or not support for Perforce Streams is enabled.

```ruby
p4 = P4.new
puts ( p4.streams? )
p4.tagged = false
puts ( p4.streams? )
```

p4.tagged( aBool ) { block }

Temporarily toggles the use of tagged output for the duration of the block, and then resets it when the block terminates.

p4.tagged= aBool -> aBool

Sets tagged output. By default, tagged output is on.

```ruby
p4 = P4.new
p4.tagged = false
```

p4.tagged? -> aBool

Detects whether or not you are in tagged mode.
p4 = P4.new
puts ( p4.tagged? )
p4.tagged = false
puts ( p4.tagged? )

**p4.ticketfile= aString -> aString**

Sets the location of the P4TICKETS file.

p4 = P4.new
p4.ticketfile = "/home/bruno/tickets"

**p4.ticketfile -> aString**

Get the path to the current P4TICKETS file.

p4 = P4.new
puts( p4.ticketfile )

**p4.track= -> aBool**

Instruct the server to return messages containing performance tracking information. By default, server tracking is disabled.

p4 = P4.new
p4.track = true

**p4.track? -> aBool**

Detects whether or not performance tracking is enabled.

p4 = P4.new
p4.track = true
puts ( p4.track? )
p4.track = false
puts ( p4.track? )

**p4.track_output -> anArray**

If performance tracking is enabled with p4.track=, returns a list of strings corresponding to the performance tracking output for the most recently-executed command.
p4 = P4.new
p4.track = true
p4.run_info
puts ( p4.track_output[0].slice(0,3) )  # should be "rpc"

**p4.user= aString -> aString**

Set the Perforce username. If not called, defaults to the value of **P4USER** taken from any **P4CONFIG** file present, or from the environment as per the usual Perforce convention. Must be called before connecting to the Perforce server.

```
p4 = P4.new
p4.user = "bruno"
p4.connect
...
p4.disconnect
```

**p4.user -> aString**

Returns the current Perforce username.

```
p4 = P4.new
puts( p4.user )
```

**p4.version= aString -> aString**

Set the version of your script, as reported to the Perforce Server.

**p4.version -> aString**

Get the version of your script, as reported to the Perforce Server.

**p4.warnings -> anArray**

Returns the array of warnings that arose during execution of the last command.

```
p4 = P4.new
begin
  p4.connect
  p4.exception_level( P4::RAISE_ALL ) # File(s) up-to-date is a warning
  files = p4.run_sync
rescue P4Exception => ex
  p4.warnings.each { |w| puts( w ) }
ensure
  p4.disconnect
end
```
Class P4Exception

Shallow subclass of `RuntimeError` to be used for catching Perforce-specific errors. Doesn’t contain any extra information. See `P4#errors()` and `P4#warnings` for details of the errors giving rise to the exception.

Class Methods

None.

Instance Methods

None.
Class P4::DepotFile

Description
Utility class providing easy access to the attributes of a file in a Perforce depot. Each P4::DepotFile object contains summary information about the file, and a list of revisions (P4::Revision objects) of that file. Currently, only the P4#run_filelog() method returns an array of P4::DepotFile objects.

Class Methods
None.

Instance Methods

df.depot_file -> aString
Returns the name of the depot file to which this object refers.

df.each_revision { |rev| block } -> revArray
Iterates over each revision of the depot file.

df.revisions -> aArray
Returns an array of revisions of the depot file.
Class P4::Revision

Description

Utility class providing easy access to the revisions of a file in a Perforce depot. P4::Revision objects can store basic information about revisions and a list of the integrations for that revision. Created by P4#run_filelog().

Class Methods

None.

Instance Methods

rev.action -> aString

Returns the name of the action which gave rise to this revision of the file.

rev.change -> aNumber

Returns the change number that gave rise to this revision of the file.

rev.client -> aString

Returns the name of the client from which this revision was submitted.

rev.depot_file -> aString

Returns the name of the depot file to which this object refers.

rev.desc -> aString

Returns the description of the change which created this revision. Note that only the first 31 characters are returned unless you use p4 filelog -L for the first 250 characters, or p4 filelog -l for the full text.

rev.digest -> aString

Returns the MD5 digest for this revision of the file.

rev.each_integration { |integ| block } -> integArray

Iterates over each the integration records for this revision of the depot file.

rev.filesize -> aNumber

Returns size of this revision.
rev.integrations -> integArray

Returns the list of integrations for this revision.

rev.rev -> aNumber

Returns the number of this revision of the file.

rev.time -> aTime

Returns the date/time that this revision was created.

rev.type -> aString

Returns this revision’s Perforce filetype.

rev.user -> aString

Returns the name of the user who created this revision.
Class P4::Integration

Description

Utility class providing easy access to the details of an integration record. Created by `P4#run_filelog()`.

Class Methods

None.

Instance Methods

`integ.how -> aString`

Returns the type of the integration record - how that record was created.

`integ.file -> aPath`

Returns the path to the file being integrated to/from.

`integ.srev -> aNumber`

Returns the start revision number used for this integration.

`integ.erev -> aNumber`

Returns the end revision number used for this integration.
Class P4::Map

Description

The P4::Map class allows users to create and work with Perforce mappings, without requiring a connection to a Perforce server.

Class Methods

Map.new ([anArray]) -> aMap

Constructs a new P4::Map object.

Map.join (map1, map2) -> aMap

Join two P4::Map objects and create a third.

The new map is composed of the left-hand side of the first mapping, as joined to the right-hand side of the second mapping. For example:

```ruby
# Map depot syntax to client syntax
client_map = P4::Map.new
client_map.insert( "//depot/main/...", "//client/..." )

# Map client syntax to local syntax
client_root = P4::Map.new
client_root.insert( "//client/...", "//home/bruno/workspace/..." )

# Join the previous mappings to map depot syntax to local syntax
local_map = P4::Map.join( client_map, client_root )
local_path = local_map.translate( "//depot/main/www/index.html" )

# local_path is now /home/bruno/workspace/www/index.html
```

Instance Methods

map.clear -> true

Empty a map.

map.count -> anInteger

Return the number of entries in a map.

map.empty? -> aBool

Test whether a map object is empty.
map.insert( aString, [ aString ] ) -> aMap

Inserts an entry into the map.

May be called with one or two arguments. If called with one argument, the string is assumed to be a string containing either a half-map, or a string containing both halves of the mapping. In this form, mappings with embedded spaces must be quoted. If called with two arguments, each argument is assumed to be half of the mapping, and quotes are optional.

# called with two arguments:
map.insert( "//depot/main/…", "//client/…" )

# called with one argument containing both halves of the mapping:
map.insert( "//depot/live/… //client/live/…" )

# called with one argument containing a half-map:
# This call produces the mapping "depot/… depot/…"
map.insert( "depot/…" )

map.translate ( aString, [ aBool ] ) -> aString

Translate a string through a map, and return the result. If the optional second argument is true, translate forward, and if it is false, translate in the reverse direction. By default, translation is in the forward direction.

map.includes? ( aString ) -> aBool

Tests whether a path is mapped or not.

if( map.includes?( "//depot/main/…" ) )
  …
end

map.reverse -> aMap

Return a new P4::Map object with the left and right sides of the mapping swapped. The original object is unchanged.

map.lhs -> anArray

Returns the left side of a mapping as an array.

map.rhs -> anArray

Returns the right side of a mapping as an array.

map.to_a -> anArray

Returns the map as an array.
Class P4::MergeData

Description

Class containing the context for an individual merge during execution of a `p4 resolve`.

Class Methods

None.

Instance Methods

\texttt{md.your\_name()} -> \texttt{aString}

Returns the name of "your" file in the merge. This is typically a path to a file in the workspace.

\begin{verbatim}
\texttt{p4.run\_resolve() do}
  \texttt{|md|}
  \texttt{yours = md.your\_name}
  \texttt{md.merge\_hint # merge result}
\texttt{end}
\end{verbatim}

\texttt{md.their\_name()} -> \texttt{aString}

Returns the name of "their" file in the merge. This is typically a path to a file in the depot.

\begin{verbatim}
\texttt{p4.run\_resolve() do}
  \texttt{|md|}
  \texttt{theirs = md.their\_name}
  \texttt{md.merge\_hint # merge result}
\texttt{end}
\end{verbatim}

\texttt{md.base\_name()} -> \texttt{aString}

Returns the name of the "base" file in the merge. This is typically a path to a file in the depot.

\begin{verbatim}
\texttt{p4.run\_resolve() do}
  \texttt{|md|}
  \texttt{base = md.base\_name}
  \texttt{md.merge\_hint # merge result}
\texttt{end}
\end{verbatim}

\texttt{md.your\_path()} -> \texttt{aString}

Returns the path of "your" file in the merge. This is typically a path to a file in the workspace.
p4.run_resolve() do
|md|
  your_path = md.your_path
  md.merge_hint # merge result
end

**md.their_path() -> aString**

Returns the path of "their" file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `P4::MergeData#their_name()` have been loaded.

p4.run_resolve() do
|md|
  their_name = md.their_name
  their_file = File.open( md.their_path )
  md.merge_hint # merge result
end

**md.base_path() -> aString**

Returns the path of the base file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `P4::MergeData#base_name()` have been loaded.

p4.run_resolve() do
|md|
  base_name = md.base_name
  base_file = File.open( md.base_path )
  md.merge_hint # merge result
end

**md.result_path() -> aString**

Returns the path to the merge result. This is typically a path to a temporary file on your local machine in which the contents of the automatic merge performed by the server have been loaded.

p4.run_resolve() do
|md|
  result_file = File.open( md.result_path )
  md.merge_hint # merge result
end

**md.merge_hint() -> aString**

Returns the hint from the server as to how it thinks you might best resolve this merge.
p4.run_resolve() do
  |md|
  puts ( md.merge_hint ) # merge result
end

md.run_merge() -> aBool

If the environment variable P4MERGE is defined, P4::MergeData#run_merge() invokes the specified program and returns a boolean based on the return value of that program.

p4.run_resolve() do
  |md|
  if ( md.run_merge() )
    "am"
  else
    "s"
  end
end
Class P4::Message

Description

P4::Message objects contain error or other diagnostic messages from the Perforce Server; retrieve them by using the P4#messages() method.

Script writers can test the severity of the messages in order to determine if the server message consisted of command output (E_INFO), warnings, (E_WARN), or errors (E_FAILED/E_FATAL).

Class methods

None.

Instance methods

message.severity() -> anInteger

Severity of the message, which is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_EMPTY</td>
<td>No error</td>
</tr>
<tr>
<td>E_INFO</td>
<td>Informational message only</td>
</tr>
<tr>
<td>E_WARN</td>
<td>Warning message only</td>
</tr>
<tr>
<td>E_FAILED</td>
<td>Command failed</td>
</tr>
<tr>
<td>E_FATAL</td>
<td>Severe error; cannot continue.</td>
</tr>
</tbody>
</table>

message.generic() -> anInteger

Returns the generic class of the error.

message.msgid() -> anInteger

Returns the unique ID of the message.

message.to_s() -> aString

Converts the message into a string.

message.inspect() -> aString

To facilitate debugging, returns a string that holds a formatted representation of the entire P4::Message object.
Class P4::OutputHandler

Description

The P4::OutputHandler class is a handler class that provides access to streaming output from the server. After defining the output handler, set P4#handler() to an instance of a subclass of P4::OutputHandler (or use a p4.with_handler( handler ) block) to enable callbacks.

By default, P4::OutputHandler returns P4::REPORT for all output methods. The different return options are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4::REPORT</td>
<td>Messages added to output.</td>
</tr>
<tr>
<td>P4::HANDLED</td>
<td>Output is handled by class (don't add message to output).</td>
</tr>
<tr>
<td>P4::CANCEL</td>
<td>Operation is marked for cancel, message is added to output.</td>
</tr>
</tbody>
</table>

Class Methods

new P4::MyHandler.new -> aP4::OutputHandler

Constructs a new subclass of P4::OutputHandler.

Instance Methods

outputBinary -> int

Process binary data.

outputInfo -> int

Process tabular data.

outputMessage -> int

Process informational or error messages.

outputStat -> int

Process tagged data.

outputText -> int

Process text data.
Class P4::Progress

Description

The P4::Progress class is a handler class that provides access to progress indicators from the server. After defining the output handler, set P4#progress() to an instance of a subclass of P4::Progress (or use a p4.with_progress( progress ) block) to enable callbacks.

You must implement all five of the following methods: init(), description(), update(), total(), and done(), even if the implementation consists of trivially returning 0.

Class Methods

new P4::MyProgress.new -> aP4::Progress

Constructs a new subclass of P4::Progress.

Instance Methods

init -> int

Initialize progress indicator.

description -> int

Description and type of units to be used for progress reporting.

update -> int

If non-zero, user has requested a cancellation of the operation.

total -> int

Total number of units expected (if known).

done -> int

If non-zero, operation has failed.
Class P4::Spec

Description

The P4::Spec class is a hash containing key/value pairs for all the fields in a Perforce form. It provides two things over and above its parent class (Hash):

- Fieldname validation. Only valid field names may be set in a P4::Spec object. Note that only the field name is validated, not the content.
- Accessor methods for easy access to the fields.

Class Methods

new P4::Spec.new( anArray ) -> aP4::Spec

Constructs a new P4::Spec object given an array of valid fieldnames.

Instance Methods

spec._<fieldname> -> aValue

Returns the value associated with the field named <fieldname>. This is equivalent to spec[ "<fieldname>" ] with the exception that when used as a method, the fieldnames may be in lowercase regardless of the actual case of the fieldname.

```ruby
client = p4.fetch_client()
root   = client._root
desc   = client._description
```

spec._<fieldname>= aValue -> aValue

Updates the value of the named field in the spec. Raises a P4Exception if the fieldname is not valid for specs of this type.

```ruby
client = p4.fetch_client()
client._root = "/home/bruno/new-client"
client._description = "My new client spec"
p4.save_client( client )
```

spec.permitted_fields -> anArray

Returns an array containing the names of fields that are valid in this spec object. This does not imply that values for all of these fields are actually set in this object, merely that you may choose to set values for any of these fields if you want to.
client = p4.fetch_client()
spec.permitted_fields.each do
  | field |
  printf ( "%14s = %s\n", field, client[ field ] )
end
Introduction

P4Perl is a Perl module that provides an object-oriented API to the Perforce version management system. Using P4Perl is faster than using the command-line interface in scripts, because multiple command can be executed on a single connection, and because it returns the Perforce Server’s responses as Perl hashes and arrays.

The main features are:

- Get Perforce data and forms in hashes and arrays.
- Edit Perforce forms by modifying hashes.
- Run as many commands on a connection as required.
- The output of commands is returned as a Perl array.
- The elements of the array returned are strings or, where appropriate, hash references.

The most recent release of P4Perl is 2014.1.

System Requirements

P4Perl is supported on Windows, Linux, Solaris, OS X, and FreeBSD. To build P4Perl, your development machine must also have:

- Perl 5.12, 5.14, or 5.16 (ActivePerl on Windows) development files.
- `make` (or `nmake` on Windows)
- The 2014.1 Perforce C/C++ API for your target platform. Older releases might work, but are not supported.
- The same C++ compiler used to build the Perforce C++ API on your target platform.

(If you get “unresolved symbol” errors when building or running P4Perl, you probably used the wrong compiler or the wrong Perforce API build.)

Installing P4Perl

You can download P4Perl from the Perforce web site:

http://www.perforce.com/product/components/apis

After downloading, you can either run the installer or build the interface from source, as described in the release notes.

Programming with P4Perl

The following example shows how to connect to a Perforce server, run a `p4 info` command, and open a file for edit.
use P4;
my $p4 = new P4;
$p4->SetClient( $clientname );
$p4->SetPort( $p4port );
$p4->SetPassword( $p4password );
$p4->Connect()
or die( "Failed to connect to Perforce Server" );

my $info = $p4->Run( "info" );
$p4->RunEdit( "file.txt" );
die( "Failed to edit file.txt" )
  if $p4->ErrorCount()
or $p4->WarningCount;

$p4->Disconnect();

Connecting to Perforce over SSL

Scripts written with P4Perl use any existing P4TRUST file present in their operating environment (by default, .p4trust in the home directory of the user that runs the script).

If the fingerprint returned by the server fails to match the one installed in the P4TRUST file associated with the script's run-time environment, your script will (and should!) fail to connect to the server.

P4Perl Classes

The P4 module consists of several public classes:

- “P4” on page 51
- “P4::DepotFile” on page 54
- “P4::Revision” on page 54
- “P4::Integration” on page 55
- “P4::Map” on page 55
- “P4::MergeData” on page 56
- “P4::Message” on page 57
- “P4::OutputHandler” on page 57
- “P4::Progress” on page 57
- “P4::Spec” on page 58

The following tables provide brief details about each public class.
The main class used for executing Perforce commands. Almost everything you do with P4Perl will involve this class.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new()</code></td>
<td>Construct a new P4 object.</td>
</tr>
<tr>
<td><code>Identify()</code></td>
<td>Print build information including P4Perl version and Perforce API version.</td>
</tr>
<tr>
<td><code>ClearHandler()</code></td>
<td>Clear the output handler.</td>
</tr>
<tr>
<td><code>Connect()</code></td>
<td>Initialize the Perforce client and connect to the Server.</td>
</tr>
<tr>
<td><code>Disconnect()</code></td>
<td>Disconnect from the Perforce Server.</td>
</tr>
<tr>
<td><code>ErrorCount()</code></td>
<td>Returns the number of errors encountered during execution of the last command.</td>
</tr>
<tr>
<td><code>Errors()</code></td>
<td>Returns a list of the error strings received during execution of the last command.</td>
</tr>
<tr>
<td><code>Fetch&lt;spectype&gt;()</code></td>
<td>Shorthand for running:</td>
</tr>
<tr>
<td></td>
<td><code>$p4-&gt;Run(&quot;&lt;spectype&gt;&quot;, &quot;-o&quot; );</code></td>
</tr>
<tr>
<td><code>Format&lt;spectype&gt;()</code></td>
<td>Shorthand for running:</td>
</tr>
<tr>
<td></td>
<td><code>$p4-&gt;FormatSpec(&quot;&lt;spectype&gt;&quot;, hash );</code></td>
</tr>
<tr>
<td><code>FormatSpec()</code></td>
<td>Converts a Perforce form of the specified type (client/label etc.) held in the supplied hash into its string representation.</td>
</tr>
<tr>
<td><code>GetApiLevel()</code></td>
<td>Get current API compatibility level.</td>
</tr>
<tr>
<td><code>GetCharset()</code></td>
<td>Get character set when connecting to Unicode servers.</td>
</tr>
<tr>
<td><code>GetClient()</code></td>
<td>Get current client workspace (P4CLIENT).</td>
</tr>
<tr>
<td><code>GetCwd()</code></td>
<td>Get current working directory.</td>
</tr>
<tr>
<td><code>GetEnv()</code></td>
<td>Get the value of a Perforce environment variable, taking into account P4CONFIG files and (on Windows or OS X) the registry or user preferences.</td>
</tr>
<tr>
<td><code>GetHandler()</code></td>
<td>Get the output handler.</td>
</tr>
<tr>
<td><code>GetHost()</code></td>
<td>Get the current hostname.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetMaxLockTime()</td>
<td>Get MaxLockTime used for all following commands.</td>
</tr>
<tr>
<td>GetMaxResults()</td>
<td>Get MaxResults used for all following commands.</td>
</tr>
<tr>
<td>GetMaxScanRows()</td>
<td>Get MaxScanRows used for all following commands.</td>
</tr>
<tr>
<td>GetPassword()</td>
<td>Get the current password or ticket.</td>
</tr>
<tr>
<td>GetPort()</td>
<td>Get host and port (P4PORT).</td>
</tr>
<tr>
<td>GetProg()</td>
<td>Get the program name as shown by the p4 monitor show -e command.</td>
</tr>
<tr>
<td>GetProgress()</td>
<td>Get the progress indicator.</td>
</tr>
<tr>
<td>GetTicketFile()</td>
<td>Get the location of the P4TICKETS file.</td>
</tr>
<tr>
<td>GetUser()</td>
<td>Get the current username (P4USER).</td>
</tr>
<tr>
<td>GetVersion()</td>
<td>Get the version of your script, as reported to the Perforce Server.</td>
</tr>
<tr>
<td>IsConnected()</td>
<td>Test whether or not session has been connected and/or has been dropped.</td>
</tr>
<tr>
<td>IsStreams()</td>
<td>Test whether or not streams are enabled.</td>
</tr>
<tr>
<td>IsTagged()</td>
<td>Test whether or not tagged output is enabled.</td>
</tr>
<tr>
<td>IsTrack()</td>
<td>Test whether or not server performance tracking is enabled.</td>
</tr>
<tr>
<td>Iterate&lt;SpecType&gt;()</td>
<td>Iterate through spec results.</td>
</tr>
<tr>
<td>Messages()</td>
<td>Return an array of P4::Message objects, one for each message sent by the server.</td>
</tr>
<tr>
<td>P4ConfigFile()</td>
<td>Get the location of the configuration file used (P4CONFIG).</td>
</tr>
<tr>
<td>Parse&lt;SpecType&gt;()</td>
<td>Shorthand for running:</td>
</tr>
<tr>
<td></td>
<td><code>$p4-ParseSpec( &quot;&lt;spectype&gt;&quot;, buffer );</code></td>
</tr>
<tr>
<td>ParseSpec()</td>
<td>Converts a Perforce form of the specified type (client, label, etc.) held in the supplied string into a hash and returns a reference to that hash.</td>
</tr>
<tr>
<td>RunCmd()</td>
<td>Shorthand for running:</td>
</tr>
<tr>
<td></td>
<td><code>$p4-Run( &quot;cmd&quot;, arg, ... );</code></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Run()</td>
<td>Run a Perforce command and return its results. Check for errors with P4::ErrorCount().</td>
</tr>
<tr>
<td>RunFilelog()</td>
<td>Runs a p4 filelog on the fileSpec provided and returns an array of P4::DepotFile objects.</td>
</tr>
<tr>
<td>RunLogin()</td>
<td>Runs p4 login using a password or ticket set by the user.</td>
</tr>
<tr>
<td>RunPassword()</td>
<td>A thin wrapper for changing your password.</td>
</tr>
<tr>
<td>RunResolve()</td>
<td>Interface to p4 resolve.</td>
</tr>
<tr>
<td>RunSubmit()</td>
<td>Submit a changelist to the server.</td>
</tr>
<tr>
<td>RunTickets()</td>
<td>Get a list of tickets from the local tickets file.</td>
</tr>
<tr>
<td>Save$spectype()</td>
<td>Shorthand for running: $p4-&gt;SetInput( $spectype ); $p4-&gt;Run( &quot;$spectype&quot;, &quot;-i&quot; );</td>
</tr>
<tr>
<td>ServerCaseSensitive()</td>
<td>Returns an integer specifying whether or not the server is case-sensitive.</td>
</tr>
<tr>
<td>ServerLevel()</td>
<td>Returns an integer specifying the server protocol level.</td>
</tr>
<tr>
<td>ServerUnicode()</td>
<td>Returns an integer specifying whether or not the server is in Unicode mode.</td>
</tr>
<tr>
<td>SetApiLevel()</td>
<td>Specify the API compatibility level to use for this script.</td>
</tr>
<tr>
<td>SetCharset()</td>
<td>Set character set when connecting to Unicode servers.</td>
</tr>
<tr>
<td>SetClient()</td>
<td>Set current client workspace (P4CLIENT).</td>
</tr>
<tr>
<td>SetCwd()</td>
<td>Set current working directory.</td>
</tr>
<tr>
<td>SetEnv()</td>
<td>On Windows or OS X, set an environment variable in the registry or user preferences.</td>
</tr>
<tr>
<td>SetHandler()</td>
<td>Set the output handler.</td>
</tr>
<tr>
<td>SetHost()</td>
<td>Set the name of the current host (P4HOST).</td>
</tr>
<tr>
<td>SetInput()</td>
<td>Save the supplied argument as input to be supplied to a subsequent command.</td>
</tr>
<tr>
<td>SetMaxLockTime()</td>
<td>Set MaxLockTime used for all following commands.</td>
</tr>
<tr>
<td>SetMaxResults()</td>
<td>Set MaxResults used for all following commands.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>SetMaxScanRows()</code></td>
<td>Set <code>MaxScanRows</code> used for all following commands.</td>
</tr>
<tr>
<td><code>SetPassword()</code></td>
<td>Set Perforce password (<code>P4PASSWD</code>).</td>
</tr>
<tr>
<td><code>SetPort()</code></td>
<td>Set host and port (<code>P4PORT</code>).</td>
</tr>
<tr>
<td><code>SetProg()</code></td>
<td>Set the program name as shown by the <code>p4 monitor show -e</code> command.</td>
</tr>
<tr>
<td><code>SetProgress()</code></td>
<td>Set the progress indicator.</td>
</tr>
<tr>
<td><code>SetStreams()</code></td>
<td>Enable or disable streams support.</td>
</tr>
<tr>
<td><code>SetTicketFile()</code></td>
<td>Set the location of the <code>P4TICKETS</code> file.</td>
</tr>
<tr>
<td><code>SetTrack()</code></td>
<td>Activate or deactivate server performance tracking. By default, tracking is off (0).</td>
</tr>
<tr>
<td><code>SetUser()</code></td>
<td>Set the Perforce username (<code>P4USER</code>).</td>
</tr>
<tr>
<td><code>SetVersion()</code></td>
<td>Set the version of your script, as reported to the Perforce Server.</td>
</tr>
<tr>
<td><code>Tagged()</code></td>
<td>Toggles tagged output (1 or 0). By default, tagged output is on (1).</td>
</tr>
<tr>
<td><code>TrackOutput()</code></td>
<td>If performance tracking is enabled with <code>SetTrack()</code> returns an array of strings with tracking output.</td>
</tr>
<tr>
<td><code>WarningCount()</code></td>
<td>Returns the number of warnings issued by the last command.</td>
</tr>
<tr>
<td><code>Warnings()</code></td>
<td>Returns a list of the warning strings received during execution of the last command.</td>
</tr>
</tbody>
</table>

**P4::DepotFile**

Utility class allowing access to the attributes of a file in the depot. Returned by `P4::RunFilelog()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DepotFile()</code></td>
<td>Name of the depot file to which this object refers.</td>
</tr>
<tr>
<td><code>Revisions()</code></td>
<td>Returns an array of revision objects for the depot file.</td>
</tr>
</tbody>
</table>

**P4::Revision**

Utility class allowing access to the attributes of a revision of a file in the depot. Returned by `P4::RunFilelog()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Action()</code></td>
<td>Returns the action that created the revision.</td>
</tr>
</tbody>
</table>
### Chapter 3. P4Perl

#### 55 APIs for Scripting

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Change()</strong></td>
<td>Returns the changelist number that gave rise to this revision of the file.</td>
</tr>
<tr>
<td><strong>Client()</strong></td>
<td>Returns the name of the client from which this revision was submitted.</td>
</tr>
<tr>
<td><strong>DepotFile()</strong></td>
<td>Returns the name of the depot file to which this object refers.</td>
</tr>
<tr>
<td><strong>Desc()</strong></td>
<td>Returns the description of the change which created this revision.</td>
</tr>
<tr>
<td><strong>Digest()</strong></td>
<td>Returns the MD5 digest for this revision.</td>
</tr>
<tr>
<td><strong>FileSize()</strong></td>
<td>Returns the size of this revision.</td>
</tr>
<tr>
<td><strong>Integrations()</strong></td>
<td>Returns an array of P4::Integration objects representing all integration records for this revision.</td>
</tr>
<tr>
<td><strong>Rev()</strong></td>
<td>Returns the number of this revision.</td>
</tr>
<tr>
<td><strong>Time()</strong></td>
<td>Returns date/time this revision was created.</td>
</tr>
<tr>
<td><strong>Type()</strong></td>
<td>Returns the Perforce filetype of this revision.</td>
</tr>
<tr>
<td><strong>User()</strong></td>
<td>Returns the name of the user who created this revision.</td>
</tr>
</tbody>
</table>

#### P4::Integration

Utility class allowing access to the attributes of an integration record for a revision of a file in the depot. Returned by P4::RunFilelog().

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How()</strong></td>
<td>Integration method (merge/branch/copy/ignored).</td>
</tr>
<tr>
<td><strong>File()</strong></td>
<td>Integrated file.</td>
</tr>
<tr>
<td><strong>SRev()</strong></td>
<td>Start revision.</td>
</tr>
<tr>
<td><strong>ERev()</strong></td>
<td>End revision.</td>
</tr>
</tbody>
</table>

#### P4::Map

A class that allows users to create and work with Perforce mappings without requiring a connection to the Perforce Server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New()</strong></td>
<td>Construct a new Map object (class method).</td>
</tr>
<tr>
<td><strong>Join()</strong></td>
<td>Joins two maps to create a third (class method).</td>
</tr>
</tbody>
</table>
## P4::PerIPerL

Class encapsulating the context of an individual merge during execution of a `p4 resolve` command. Passed to `P4::RunResolve`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clear()</strong></td>
<td>Empties a map.</td>
</tr>
<tr>
<td><strong>Count()</strong></td>
<td>Returns the number of entries in a map.</td>
</tr>
<tr>
<td><strong>IsEmpty()</strong></td>
<td>Tests whether or not a map object is empty.</td>
</tr>
<tr>
<td><strong>Insert()</strong></td>
<td>Inserts an entry into the map.</td>
</tr>
<tr>
<td><strong>Translate()</strong></td>
<td>Translate a string through a map.</td>
</tr>
<tr>
<td><strong>Includes()</strong></td>
<td>Tests whether a path is mapped.</td>
</tr>
<tr>
<td><strong>Reverse()</strong></td>
<td>Returns a new mapping with the left and right sides reversed.</td>
</tr>
<tr>
<td><strong>Lhs()</strong></td>
<td>Returns the left side as an array.</td>
</tr>
<tr>
<td><strong>Rhs()</strong></td>
<td>Returns the right side as an array.</td>
</tr>
<tr>
<td><strong>AsArray()</strong></td>
<td>Returns the map as an array.</td>
</tr>
</tbody>
</table>

## P4::MergeData

Class encapsulating the context of an individual merge during execution of a `p4 resolve` command. Passed to `P4::RunResolve`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>YourName()</strong></td>
<td>Returns the name of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td><strong>TheirName()</strong></td>
<td>Returns the name of &quot;their&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td><strong>BaseName()</strong></td>
<td>Returns the name of &quot;base&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td><strong>YourPath()</strong></td>
<td>Returns the path of &quot;your&quot; file in the merge. (file in depot)</td>
</tr>
<tr>
<td><strong>TheirPath()</strong></td>
<td>Returns the path of &quot;their&quot; file in the merge. (temporary file on workstation into which <code>TheirName()</code> has been loaded)</td>
</tr>
<tr>
<td><strong>BasePath()</strong></td>
<td>Returns the path of the base file in the merge. (temporary file on workstation into which <code>BaseName()</code> has been loaded)</td>
</tr>
<tr>
<td><strong>ResultPath()</strong></td>
<td>Returns the path to the merge result. (temporary file on workstation into which the automatic merge performed by the server has been loaded)</td>
</tr>
<tr>
<td><strong>MergeHint()</strong></td>
<td>Returns hint from server as to how user might best resolve merge.</td>
</tr>
<tr>
<td><strong>RunMergeTool()</strong></td>
<td>If the environment variable <code>P4MERGE</code> is defined, run it and indicate whether or not the merge tool successfully executed.</td>
</tr>
</tbody>
</table>

---

APIs for Scripting
**P4::Message**

Class encapsulating the context of an individual error during execution of Perforce commands. Passed to `P4::Messages()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSeverity()</td>
<td>Returns the severity class of the error.</td>
</tr>
<tr>
<td>GetGeneric()</td>
<td>Returns the generic class of the error message.</td>
</tr>
<tr>
<td>GetId()</td>
<td>Returns the unique ID of the error message.</td>
</tr>
<tr>
<td>GetText()</td>
<td>Get the text of the error message.</td>
</tr>
</tbody>
</table>

**P4::OutputHandler**

Handler class that provides access to streaming output from the server; call `P4::SetHandler()` with an implementation of `P4::OutputHandler` to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputBinary()</td>
<td>Process binary data.</td>
</tr>
<tr>
<td>OutputInfo()</td>
<td>Process tabular data.</td>
</tr>
<tr>
<td>OutputMessage()</td>
<td>Process information or errors.</td>
</tr>
<tr>
<td>OutputStat()</td>
<td>Process tagged output.</td>
</tr>
<tr>
<td>OutputText()</td>
<td>Process text data.</td>
</tr>
</tbody>
</table>

**P4::Progress**

Handler class that provides access to progress indicators from the server; call `P4::SetProgress()` with an implementation of `P4::Progress` to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init()</td>
<td>Initialize progress indicator as designated type.</td>
</tr>
<tr>
<td>Total()</td>
<td>Total number of units (if known).</td>
</tr>
<tr>
<td>Description()</td>
<td>Description and type of units to be used for progress reporting.</td>
</tr>
<tr>
<td>Update()</td>
<td>If non-zero, user has requested a cancellation of the operation.</td>
</tr>
<tr>
<td>Done()</td>
<td>If non-zero, operation has failed.</td>
</tr>
</tbody>
</table>
**P4::Resolver**

Class for handling resolves in Perforce.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolve()</td>
<td>Perform a resolve and return the resolve decision as a string.</td>
</tr>
</tbody>
</table>

**P4::Spec**

Utility class allowing access to the attributes of the fields in a Perforce form.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_fieldname()</td>
<td>Return the value associated with the field named <em>fieldname</em>.</td>
</tr>
<tr>
<td>_fieldname()</td>
<td>Set the value associated with the field named <em>fieldname</em>.</td>
</tr>
<tr>
<td>PermittedFields()</td>
<td>Lists the fields that are permitted for specs of this type.</td>
</tr>
</tbody>
</table>
Class P4

Description

Main interface to the Perforce client API.

This module provides an object-oriented interface to the Perforce version management system. Data is returned in Perl arrays and hashes and input can also be supplied in these formats.

Each P4 object represents a connection to the Perforce Server, and multiple commands may be executed (serially) over a single connection.

The basic model is to:

1. Instantiate your P4 object.
2. Specify your Perforce client environment.
   - SetClient()
   - SetHost()
   - SetPassword()
   - SetPort()
   - SetUser()
3. Connect to the Perforce service.

   The Perforce protocol is not designed to support multiple concurrent queries over the same connection. Multithreaded applications that use the C++ API or derived APIs (including P4Perl) should ensure that a separate connection is used for each thread, or that only one thread may use a shared connection at a time.

4. Run your Perforce commands.
5. Disconnect from the Perforce service.

Class methods

P4::new() -> P4

Construct a new P4 object. For example:

```perl
my $p4 = new P4;
```

P4::Identify() -> string

Print build information including P4Perl version and Perforce API version.
print P4::Identify();

The constants OS, PATCHLEVEL and VERSION are also available to test an installation of P4Perl without having to parse the output of P4::Identify(). Also reports the version of the OpenSSL library used for building the underlying Perforce C++ API with which P4Perl was built.

**P4::ClearHandler() -> undef**

Clear any configured output handler.

**P4::Connect() -> bool**

Initializes the Perforce client and connects to the server. Returns false on failure and true on success.

**P4::Disconnect() -> undef**

Terminate the connection and clean up. Should be called before exiting.

**P4::ErrorCount() -> integer**

Returns the number of errors encountered during execution of the last command.

**P4::Errors() -> list**

Returns a list of the error strings received during execution of the last command.

**P4::Fetch<Spectype>( [name] ) -> hashref**

Shorthand for running:

```perl
$p4->Run( "<spectype>", "-o" );
```

and returning the first element of the result array. For example:

```perl
$label = $p4->FetchLabel( $labelname );
$change = $p4->FetchChange( $changeno );
$clientspec = $p4->FetchClient( $clientname );
```

**P4::Format<Spectype>( hash ) -> string**

Shorthand for running:

```perl
$p4->FormatSpec( "<spectype>", hash );
```

and returning the results. For example:
$change = $p4->FetchChange();

$change->{ 'Description' } = 'Some description';

$form = $p4->FormatChange( $change );

printf( "Submitting this change:\n\n%s\n", $form );

$p4->RunSubmit( $change );

---

**P4::FormatSpec( $spectype, $string ) -> string**

Converts a Perforce form of the specified type (client, label, etc.) held in the supplied hash into its string representation. Shortcut methods are available that obviate the need to supply the type argument. The following two examples are equivalent:

```perl
my $client = $p4->FormatSpec( "client", $hash );

my $client = $p4->FormatClient( $hash );
```

**P4::GetApiLevel() -> integer**

Returns the current API compatibility level. Each iteration of the Perforce Server is given a level number. As part of the initial communication, the client protocol level is passed between client application and the Perforce Server. This value, defined in the Perforce API, determines the communication protocol level that the Perforce client will understand. All subsequent responses from the Perforce Server can be tailored to meet the requirements of that client protocol level.

For more information, see:

[http://kb.perforce.com/article/512](http://kb.perforce.com/article/512)

**P4::GetCharset() -> string**

Return the name of the current charset in use. Applicable only when used with Perforce servers running in unicode mode.

**P4::GetClient() -> string**

Returns the current Perforce client name. This may have previously been set by `P4::SetClient()`, or may be taken from the environment or P4CONFIG file if any. If all that fails, it will be your hostname.

**P4::GetCwd() -> string**

Returns the current working directory as your Perforce client sees it.

**P4::GetEnv( $var ) -> string**

Returns the value of a Perforce environment variable, taking into account the settings of Perforce variables in P4CONFIG files, and, on Windows or OS X, in the registry or user preferences.
**P4::GetHandler() -> Handler**

Returns the output handler.

**P4::GetHost() -> string**

Returns the client hostname. Defaults to your hostname, but can be overridden with **P4::SetHost()**.

**P4::GetMaxLockTime( $value ) -> integer**

Get the current maxlocktime setting.

**P4::GetMaxResults( $value ) -> integer**

Get the current maxresults setting.

**P4::GetMaxScanRows( $value ) -> integer**

Get the current maxscanrows setting.

**P4::GetPassword() -> string**

Returns your Perforce password. Taken from a previous call to **P4::SetPassword()** or extracted from the environment (**$ENV{P4PASSWD}**), or a **P4CONFIG** file.

**P4::GetPort() -> string**

Returns the current address for your Perforce server. Taken from a previous call to **P4::SetPort()**, or from **$ENV{P4PORT}** or a **P4CONFIG** file.

**P4::GetProg() -> string**

Get the name of the program as reported to the Perforce Server.

**P4::GetProgress() -> Progress**

Returns the progress indicator.

**P4::GetTicketFile( [ $string ] ) -> string**

Return the path of the current **P4TICKETS** file.

**P4::GetUser() -> String**

Get the current user name. Taken from a previous call to **P4::SetUser()**, or from **$ENV{P4USER}** or a **P4CONFIG** file.

**P4::GetVersion( $string ) -> string**

Get the version of your script, as reported to the Perforce Server.
Chapter 3. P4Perl

**P4::IsConnected() -> bool**

Returns true if the session has been connected, and has not been dropped.

**P4::IsStreams() -> bool**

Returns true if streams support is enabled on this server.

**P4::IsTagged() -> bool**

Returns true if Tagged mode is enabled on this client.

**P4::IsTrack() -> bool**

Returns true if server performance tracking is enabled for this connection.

**P4::Iterate<SpecType>(arguments) -> object**

Iterate over spec results. Returns an iterable object with `next()` and `hasNext()` methods.

Valid `<specType>`s are `clients`, `labels`, `branches`, `changes`, `streams`, `jobs`, `users`, `groups`, `depots` and `servers`. Valid arguments are any arguments that would be valid for the corresponding `P4::RunCmd()` command.

Arguments can be passed to the iterator to filter the results, for example, to iterate over only the first two client workspace specifications:

```perl
$p4->IterateClients( "-m2" );
```

You can also pass the spec type as an argument:

```perl
$p4->Iterate( "changes" );
```

For example, to iterate through client specs:

```perl
use P4;
my $p4 = P4->new;
$p4->Connect or die "Couldn't connect";
my $i = $p4->IterateClients();
while($i->hasNext) {
    my $spec = $i->next;
    print( "Client: ". ($spec->{Client} or "<undef>" ). "\n" );
}
```

**P4::Messages() -> list**

Returns an array of `P4::Message()` objects, one for each message (info, warning or error) sent by the server.
P4::P4ConfigFile() -> string

Get the path to the current P4CONFIG file.

P4::Parse<Spectype>( $string ) -> hashref

Shorthand for running:

```perl
$p4=ParseSpec( "<spectype>", buffer);
```

and returning the results. For example:

```perl
$p4 = new P4;
$p4->Connect() or die( "Failed to connect to server" );
$client = $p4->FetchClient();

# Returns a hashref
$client = $p4->FormatClient( $client );

# Convert to string
$client = $p4->ParseClient( $client );
# Convert back to hashref
```

Comments in forms are preserved. Comments are stored as a comment key in the spec hash and are accessible. For example:

```perl
my $spec = $pc->ParseGroup( 'my_group' );
print $spec->{'comment'};
```

P4::ParseSpec( $spectype, $string ) -> hashref

Converts a Perforce form of the specified type (client/label etc.) held in the supplied string into a hash and returns a reference to that hash. Shortcut methods are available to avoid the need to supply the type argument. The following two examples are equivalent:

```perl
my $hash = $p4->ParseSpec( "client", $clientspec );

my $hash = $p4->ParseClient( $clientspec );
```

P4::Run<Cmd>( [ $arg... ] ) -> list | arrayref

Shorthand for running:
$p4-Run("cmd", arg, ...);

and returning the results.

**P4::Run("<cmd>",[ $arg... ]) -> list | arrayref**

Run a Perforce command and return its results. Because Perforce commands can partially succeed and partially fail, it is good practice to check for errors using `P4::ErrorCount()`.

Results are returned as follows:

- A list of results in array context
- An array reference in scalar context

The AutoLoader enables you to treat Perforce commands as methods:

```perl
p4->RunEdit("filename.txt");
```

is equivalent to:

```perl
$p4->Run("edit", "filename.txt");
```

Note that the content of the array of results you get depends on (a) whether you're using tagged mode, (b) the command you've executed, (c) the arguments you supplied, and (d) your Perforce server version.

Tagged mode and form parsing mode are turned on by default; each result element is a hashref, but this is dependent on the command you ran and your server version.

In non-tagged mode, each result element is a string. In this case, because the Perforce server sometimes asks the client to write a blank line between result elements, some of these result elements can be empty.

Note that the return values of individual Perforce commands are not documented because they may vary between server releases.

To correlate the results returned by the P4 interface with those sent to the command line client, try running your command with RPC tracing enabled. For example:

Tagged mode: `p4 -Ztag -vrpc=1 describe -s 4321`

Non-Tagged mode: `p4 -vrpc=1 describe -s 4321`

FIXME Pay attention to the calls to `client-FstatInfo()`, `client-OutputText()`, `client-OutputData()` and `client-HandleError()`. Each call to one of these functions results in either a result element, or an error element.
**P4::RunFilelog([$args ...], $fileSpec ...) -> list | arrayref**

Runs a `p4 filelog` on the $fileSpec provided and returns an array of `P4::DepotFile` objects when executed in tagged mode.

**P4::RunLogin(...) -> list | arrayref**

Runs `p4 login` using a password or ticket set by the user.

**P4::RunPassword($oldpass, $newpass) -> list | arrayref**

A thin wrapper for changing your password from `$oldpass` to `$newpass`. Not to be confused with `P4::SetPassword()`.

**P4::RunResolve([$resolver], [$args ...]) -> string**

Run a `p4 resolve` command. Interactive resolves require the `$resolver` parameter to be an object of a class derived from `P4::Resolver`. In these cases, the `P4::Resolve()` method of this class is called to handle the resolve. For example:

```
$resolver = new MyResolver;
$p4->RunResolve($resolver);
```

To perform an automated merge that skips whenever conflicts are detected:

```
use P4;

package MyResolver;
our @ISA = qw( P4::Resolver );

sub Resolve( $ ) {
    my $self = shift;
    my $mergeData = shift;

    # "s"kip if server-recommended hint is to "e"dit the file,
    # because such a recommendation implies the existence of a conflict
    return "s" if ( $mergeData->Hint() eq "e" );
    return $mergeData->Hint();
}
1;

package main;

$p4 = new P4;
$resolver = new MyResolver;

$p4->Connect() or die( "Failed to connect to Perforce" );
$p4->RunResolve($resolver, ...);
```

In non-interactive resolves, no `P4::Resolver` object is required. For example:
$p4->RunResolve( "at" );

**P4::RunSubmit**($arg | $hashref, ...) -> list | arrayref

Submit a changelist to the server. To submit a changelist, set the fields of the changelist as required and supply any flags:

```
$change = $p4->FetchChange();
$change->{ 'Description' } = "Some description";
$p4->RunSubmit( "-r", $change );
```

You can also submit a changelist by supplying the arguments as you would on the command line:

```
$p4->RunSubmit( "-d", "Some description", "somedir/..." );
```

**P4::RunTickets()** -> list

Get a list of tickets from the local tickets file. Each ticket is a hash object with fields for Host, User, and Ticket.

**P4::Save<Spectype>()** -> list | arrayref

Shorthand for running:

```
$p4->SetInput( $spectype );
$p4->Run( "<spectype>" , "-i" );
```

For example:

```
$p4->SaveLabel( $label );
$p4->SaveChange( $changeno );
$p4->SaveClient( $clientspec );
```

**P4::ServerCaseSensitive()** -> integer

Returns an integer specifying whether or not the server is case-sensitive.

**P4::ServerLevel()** -> integer

Returns an integer specifying the server protocol level. This is not the same as, but is closely aligned to, the server version. To find out your server's protocol level, run `p4 -vrpc=5 info` and look for the server2 protocol variable in the output. For more information, see:

[http://kb.perforce.com/article/571](http://kb.perforce.com/article/571)
P4::ServerUnicode() -> integer

Returns an integer specifying whether or not the server is in Unicode mode.

P4::SetApiLevel( $integer ) -> undef

Specify the API compatibility level to use for this script. This is useful when you want your script to continue to work on newer server versions, even if the new server adds tagged output to previously unsupported commands.

The additional tagged output support can change the server’s output, and confound your scripts. Setting the API level to a specific value allows you to lock the output to an older format, thus increasing the compatibility of your script.

Must be called before calling P4::Connect(). For example:

```perl
$p4->SetApiLevel( 67 ); # Lock to 2010.1 format
$p4->Connect() or die( "Failed to connect to Perforce" );
# etc.
```

P4::SetCharset( $charset ) -> undef

Specify the character set to use for local files when used with a Perforce server running in unicode mode. Do not use unless your Perforce server is in unicode mode. Must be called before calling P4::Connect(). For example:

```perl
$p4->SetCharset( "winansi" );
$p4->SetCharset( "iso8859-1" );
$p4->SetCharset( "utf8" );
# etc.
```

P4::SetClient( $client ) -> undef

Sets the name of your Perforce client workspace. If you don’t call this method, then the client workspace name will default according to the normal Perforce conventions:

1. Value from file specified by P4CONFIG
2. Value from $ENV{P4CLIENT}
3. Hostname

P4::SetCwd( $path ) -> undef

Sets the current working directory for the client.

P4::SetEnv( $var, $value ) -> undef

On Windows or OS X, set a variable in the registry or user preferences. To unset a variable, pass an empty string as the second argument. On other platforms, an exception is raised.
$p4->SetEnv( "P4CLIENT", "my_workspace" );
$p4->SetEnv( "P4CLIENT", "" );

P4::SetHandler( Handler ) -> Handler

Sets the output handler.

P4::SetHost( $hostname ) -> undef

Sets the name of the client host, overriding the actual hostname. This is equivalent to `p4 -H hostname`, and only useful when you want to run commands as if you were on another machine.

P4::SetInput( $string | $hashref | $arrayref ) -> undef

Save the supplied argument as input to be supplied to a subsequent command. The input may be a hashref, a scalar string, or an array of hashrefs or scalar strings. If you pass an array, the array will be shifted once each time the Perforce command being executed asks for user input.

P4::SetMaxLockTime( $integer ) -> undef

Limit the amount of time (in milliseconds) spent during data scans to prevent the server from locking tables for too long. Commands that take longer than the limit will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxresults` for information on the commands that support this limit.

P4::SetMaxResults( $integer ) -> undef

Limit the number of results for subsequent commands to the value specified. Perforce will abort the command if continuing would produce more than this number of results. Once set, this limit remains in force unless you remove the restriction by setting it to a value of 0.

P4::SetMaxScanRows( $integer ) -> undef

Limit the number of records Perforce will scan when processing subsequent commands to the value specified. Perforce will abort the command once this number of records has been scanned. Once set, this limit remains in force unless you remove the restriction by setting it to a value of 0.

P4::SetPassword( $password ) -> undef

Specify the password to use when authenticating this user against the Perforce Server - overrides all defaults. Not to be confused with `P4::Password()`.

P4::SetPort( $port ) -> undef

Set the port on which your Perforce server is listening. Defaults to:

1. Value from file specified by `P4CONFIG`
2. Value from `$ENV{P4PORT}`
3.  *perforce:1666*

**P4::SetProg( $program_name ) -> undef**

Set the name of your script. This value is displayed in the server log on 2004.2 or later servers.

**P4::SetProgress( Progress ) -> Progress**

Sets the progress indicator.

**P4::SetStreams( 0 | 1 ) -> undef**

Enable or disable support for streams. By default, streams support is enabled at 2011.1 or higher (`P4::GetApiLevel() >= 70`). Streams support requires a server at 2011.1 or higher. You can enable or disable support for streams both before and after connecting to the server.

**P4::SetTicketFile( [$string] ) -> string**

Set the path to the current `P4TICKETS` file (and return it).

**P4::SetTrack( 0 | 1 ) -> undef**

Enable (1) or disable (0) server performance tracking for this connection. By default, performance tracking is disabled.

**P4::SetUser( $username ) -> undef**

Set your Perforce username. Defaults to:

1. Value from file specified by `P4CONFIG`
2. Value from `C<$ENV{P4USER}>`
3. OS username

**P4::SetVersion( $version ) -> undef**

Specify the version of your script, as recorded in the Perforce server log file.

**P4::Tagged( 0 | 1 | $coderef ) -> undef**

Enable (1) or disable (0) tagged output from the server, or temporarily toggle it.

By default, tagged output is enabled, but can be disabled (or re-enabled) by calling this method. If you provide a code reference, you can run a subroutine with the tagged status toggled for the duration of that reference. For example:

```perl
my $GetChangeCounter = sub{ $p4->RunCounter('change')->[ 0 ] );
my $changeno = $p4->Tagged( 0, $GetChangeCounter );
```
When running in tagged mode, responses from commands that support tagged output will be returned in the form of a hashref. When running in non-tagged mode, responses from commands are returned in the form of strings (that is, in plain text).

**P4::TrackOutput() -> list**

If performance tracking is enabled with `P4::SetTrack()`, returns a list of strings corresponding to the performance tracking output of the most recently-executed command.

**P4::WarningCount() -> integer**

Returns the number of warnings issued by the last command.

```perl
$p4->WarningCount();
```

**P4::Warnings() -> list**

Returns a list of warning strings from the last command.

```perl
$p4->Warnings();
```
Class P4::DepotFile

Description

P4::DepotFile objects are used to present information about files in the Perforce repository. They are returned by P4::RunFilelog().

Class Methods

None.

Instance Methods

$df->DepotFile() -> string

Returns the name of the depot file to which this object refers.

$df->Revisions() -> array

Returns an array of P4::Revision objects, one for each revision of the depot file.
Class P4::Revision

Description

P4::Revision objects are represent individual revisions of files in the Perforce repository. They are returned as part of the output of P4::RunFilelog().

Class Methods

$rev->Integrations() -> array

Returns an array of P4::Integration objects representing all integration records for this revision.

Instance Methods

$rev->Action() -> string

Returns the name of the action which gave rise to this revision of the file.

$rev->Change() -> integer

Returns the changelist number that gave rise to this revision of the file.

$rev->Client() -> string

Returns the name of the client from which this revision was submitted.

$rev->DepotFile() -> string

Returns the name of the depot file to which this object refers.

$rev->Desc() -> string

Returns the description of the change which created this revision. Note that only the first 31 characters are returned unless you use p4 filelog -L for the first 250 characters, or p4 filelog -l for the full text.

$rev->Digest() -> string

Returns the MD5 digest for this revision.

$rev->FileSize() -> string

Returns the size of this revision.

$rev->Rev() -> integer

Returns the number of this revision of the file.
$rev->Time() - string

Returns the date/time that this revision was created.

$rev->Type() - string

Returns this revision’s Perforce filetype.

$rev->User() - string

Returns the name of the user who created this revision.
Class P4::Integration

Description

P4::Integration objects represent Perforce integration records. They are returned as part of the output of P4::RunFilelog().

Class Methods

None.

Instance Methods

$integ->How() -> string

Returns the type of the integration record - how that record was created.

$integ->File() -> string

Returns the path to the file being integrated to/from.

$integ->SRev() -> integer

Returns the start revision number used for this integration.

$integ->ERev() -> integer

Returns the end revision number used for this integration.
Class P4::Map

Description

The P4::Map class allows users to create and work with Perforce mappings, without requiring a connection to a Perforce server.

Class Methods

$map = new P4::Map([ array ]) -> aMap

Constructs a new P4::Map object.

$map->Join(map1, map2) -> aMap

Join two P4::Map objects and create a third.

The new map is composed of the left-hand side of the first mapping, as joined to the right-hand side of the second mapping. For example:

# Map depot syntax to client syntax
$client_map = new P4::Map;
$client_map->Insert("//depot/main/...", "/client/...");

# Map client syntax to local syntax
$client_root = new P4::Map;
$client_root->Insert("//client/...", "/home/bruno/workspace/...");

# Join the previous mappings to map depot syntax to local syntax
$local_map = P4::Map::Join( $client_map, $client_root );
$local_path = $local_map->Translate("//depot/main/www/index.html");

# $local_path is now /home/bruno/workspace/www/index.html

Instance Methods

$map->Clear() -> undef

Empty a map.

$map->Count() -> integer

Return the number of entries in a map.

$map->IsEmpty() -> bool

Test whether a map object is empty.
$map->Insert( string ... ) -> undef

Inserts an entry into the map.

May be called with one or two arguments. If called with one argument, the string is assumed to be a string containing either a half-map, or a string containing both halves of the mapping. In this form, mappings with embedded spaces must be quoted. If called with two arguments, each argument is assumed to be half of the mapping, and quotes are optional.

```perl
# called with two arguments:
$map->Insert( "//depot/main/...", "//client/..." );

# called with one argument containing both halves of the mapping:
$map->Insert( "//depot/live/... //client/live/..." );

# called with one argument containing a half-map:
# This call produces the mapping "depot/... depot/..."
$map->Insert( "depot/..." );
```

$map->Translate( string, [ bool ] ) -> string

Translate a string through a map, and return the result. If the optional second argument is 1, translate forward, and if it is 0, translate in the reverse direction. By default, translation is in the forward direction.

$map->Includes( string ) -> bool

Tests whether a path is mapped or not.

```perl
if ( $map->Includes( "//depot/main/..." ) ) {
  ...
}
```

$map->Reverse() -> aMap

Return a new P4::Map object with the left and right sides of the mapping swapped. The original object is unchanged.

$map->Lhs() -> array

Returns the left side of a mapping as an array.

$map->Rhs() -> array

Returns the right side of a mapping as an array.

$map->AsArray() -> array

Returns the map as an array.
Class P4::MergeData

Description

Class containing the context for an individual merge during execution of a `p4 resolve`. Users may not create objects of this class; they are created internally during `P4::RunResolve()`, and passed down to the `Resolve()` method of a `P4::Resolver` subclass.

Class Methods

None.

Instance Methods

$md.YourName() -> string

Returns the name of "your" file in the merge, in client syntax.

$md.TheirName() -> string

Returns the name of "their" file in the merge, in client syntax, including the revision number.

$md.BaseName() -> string

Returns the name of the "base" file in the merge, in depot syntax, including the revision number.

$md.YourPath() -> string

Returns the path of "your" file in the merge. This is typically a path to a file in the client workspace.

$md.TheirPath() -> string

Returns the path of "their" file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `P4::MergeData::TheirName()` have been loaded.

$md.BasePath() -> string

Returns the path of the base file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `P4::MergeData::BaseName()` have been loaded.

$md.ResultPath() -> string

Returns the path to the merge result. This is typically a path to a temporary file on your local machine in which the contents of the automatic merge performed by the server have been loaded.

$md.MergeHint() -> string

Returns a string containing the hint from Perforce's merge algorithm, indicating the recommended action for performing the resolve.
\$md.RunMergeTool() \to \text{ integer}

If the environment variable \texttt{P4MERGE} is defined, \texttt{P4::MergeData::RunMergeTool()} invokes the specified program and returns true if the merge tool was successfully executed, otherwise returns false.


Class P4::Message

Description

P4::Message objects contain error or other diagnostic messages from the Perforce Server; they are returned by P4::Messages().

Script writers can test the severity of the messages in order to determine if the server message consisted of command output (E_INFO), warnings, (E_WARN), or errors (E_FAILED / E_FATAL).

Class methods

None.

Instance methods

$message.GetSeverity() -> int

Severity of the message, which is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_EMPTY</td>
<td>No error.</td>
</tr>
<tr>
<td>E_INFO</td>
<td>Informational message only.</td>
</tr>
<tr>
<td>E_WARN</td>
<td>Warning message only.</td>
</tr>
<tr>
<td>E_FAILED</td>
<td>Command failed.</td>
</tr>
<tr>
<td>E_FATAL</td>
<td>Severe error; cannot continue.</td>
</tr>
</tbody>
</table>

$message.GetGeneric() -> int

Returns the generic class of the error.

$message.GetId() -> int

Returns the unique ID of the message.

$message.GetText() -> int

Converts the message into a string.
Class P4::OutputHandler

**Description**

The P4::OutputHandler class provides access to streaming output from the server. After defining the output handler, call P4::SetHandler() with your implementation of P4::OutputHandler.

Because P4Perl does not provide a template or superclass, your output handler must implement all five of the following methods: OutputMessage(), OutputText(), OutputInfo(), OutputBinary(), and OutputStat(), even if the implementation consists of trivially returning 0 (report only: don’t handle output, don’t cancel operation).

These methods must return one of the following four values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Messages added to output (don't handle, don't cancel).</td>
</tr>
<tr>
<td>1</td>
<td>Output is handled by class (don't add message to output).</td>
</tr>
<tr>
<td>2</td>
<td>Operation is marked for cancel, message is added to output.</td>
</tr>
<tr>
<td>3</td>
<td>Operation is marked for cancel, message not added to output.</td>
</tr>
</tbody>
</table>

**Class Methods**

None.

**Instance Methods**

$handler.OutputBinary() -> int

Process binary data.

$handler.OutputInfo() -> int

Process tabular data.

$handler.OutputMessage() -> int

Process informational or error messages.

$handler.OutputStat() -> int

Process tagged data.

$handler.OutputText() -> int

Process text data.
Class P4::Progress

Description

The P4::Progress provides access to progress indicators from the server. After defining the progress class, call P4::SetProgress() with your implementation of P4::Progress.

Because P4Perl does not provide a template or superclass, you must implement all five of the following methods: Init(), Description(), Update(), Total(), and Done(), even if the implementation consists of trivially returning 0.

Class Methods

None.

Instance Methods

$progress.Init() -> int

Initialize progress indicator.

$progress.Description( string, int ) -> int

Description and type of units to be used for progress reporting.

$progress.Update() -> int

If non-zero, user has requested a cancellation of the operation.

$progress.Total() -> int

Total number of units expected (if known).

$progress.Done() -> int

If non-zero, operation has failed.
Class P4::Resolver

Description

P4::Resolver is a class for handling resolves in Perforce. It is intended to be subclassed, and for subclasses to override the Resolve() method. When P4::RunResolve() is called with a P4::Resolver object, it calls the P4::Resolver::Resolve() method of the object once for each scheduled resolve.

Class Methods

None.

Instance Methods

$resolver.Resolve() -> string

Returns the resolve decision as a string. The standard Perforce resolve strings apply:

<table>
<thead>
<tr>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ay</td>
<td>Accept Yours.</td>
</tr>
<tr>
<td>at</td>
<td>Accept Theirs.</td>
</tr>
<tr>
<td>am</td>
<td>Accept Merge result.</td>
</tr>
<tr>
<td>ae</td>
<td>Accept Edited result.</td>
</tr>
<tr>
<td>s</td>
<td>Skip this merge.</td>
</tr>
<tr>
<td>q</td>
<td>Abort the merge.</td>
</tr>
</tbody>
</table>

By default, all automatic merges are accepted, and all merges with conflicts are skipped. The P4::Resolver::Resolve() method is called with a single parameter, which is a reference to a P4::MergeData object.
Class \texttt{P4::Spec}

\textbf{Description}

\texttt{P4::Spec} objects provide easy access to the attributes of the fields in a Perforce form.

The \texttt{P4::Spec} class uses Perl’s AutoLoader to simplify form manipulation. Form fields can be accessed by calling a method with the same name as the field prefixed by an underscore (_).

\textbf{Class Methods}

\texttt{$spec = \text{new \texttt{P4::Spec}}( \text{\$fieldMap} ) \rightarrow \text{array}}

Constructs a new \texttt{P4::Spec} object for a form containing the specified fields. (The object also contains a \_\texttt{fields} \_ member that stores a list of field names that are valid in forms of this type.)

\textbf{Instance Methods}

\texttt{$spec->\_\langle\texttt{fieldname}\rangle \rightarrow \text{string}}

Returns the value associated with the field named \texttt{\langle\texttt{fieldname}\rangle}.

\begin{verbatim}
$client = $p4->FetchClient( $clientname );
$client->_Root();  # Get client root
\end{verbatim}

\texttt{$spec->\_\langle\texttt{fieldname}\rangle( \text{\$string} )\rightarrow \text{string}}

Updates the value of the named field in the spec.

\begin{verbatim}
$client = $p4->FetchClient( $clientname );
$client->_Root( $newroot );  # Set client root
\end{verbatim}

\texttt{$spec->PermittedFields() \rightarrow \text{array}}

Returns an array containing the names of fields that are valid in this spec object. This does not imply that values for all of these fields are actually set in this object, merely that you may choose to set values for any of these fields if you want to.

\begin{verbatim}
my $client = $p4->FetchClient( $clientname );
my @fields = $p4->PermittedFields( $client );
foreach $field (@fields) {
    print "$field\n";
}
\end{verbatim}
P4Python, the Python interface to the Perforce API, enables you to write Python code that interacts with a Perforce server. P4Python enables your Python scripts to:

- Get Perforce data and forms in dictionaries and lists.
- Edit Perforce forms by modifying dictionaries.
- Provide exception-based error handling and optionally ignore warnings.
- Issue multiple commands on a single connection (performs better than spawning single commands and parsing the results).

The most recent release of P4Python is 2015.1.

### System Requirements

P4Python is supported on Windows, Linux, Solaris, OS X, and FreeBSD.

To build P4Python from source, your development machine must also have:

- Python 2.7 or 3.3 development files.
- The 2015.1 Perforce C/C++ API for your target platform.
- The same C++ compiler used to build the Perforce C++ API on your target platform.

(If you get "unresolved symbol" errors when building or running P4Python, you probably used the wrong compiler or the wrong Perforce API build.)

For the most up-to-date system requirements, see the P4Python release notes:

[http://www.perforce.com/perforce/r15.1/user/p4pythonnotes.txt](http://www.perforce.com/perforce/r15.1/user/p4pythonnotes.txt)

### Installing P4Python

**Important** Before installing P4Python, any previously installed versions should be uninstalled.

As of P4Python 2015.1, the recommended mechanism for installing P4Python is via `pip`. For example:

```
$ pip install p4python
```

`pip` installs binary versions of P4Python where possible, otherwise it attempts to automatically build P4Python from source.
Windows users can download an installer containing pre-built packages for P4Python from the Perforce web site:

http://www.perforce.com/product/components/apis

**Note**

When P4Python is built without the **--apidir** option, setup attempts to connect to ftp.perforce.com to download the correct version of the P4API binary. If the P4API download is successful, it is unpacked into a temporary directory.

When P4Python is built and the **--ssl** is provided without a path, setup attempts to determine the correct path of the installed OpenSSL libraries by executing openssl version.

### Programming with P4Python

P4Python provides an object-oriented interface to Perforce that is intended to be intuitive for Python programmers. Data is loaded and returned in Python arrays and dictionaries. Each P4 object represents a connection to the Perforce server.

When instantiated, the P4 instance is set up with the default environment settings just as the command line client p4, that is, using environment variables, the registry or user preferences (on Windows and OS X) and, if defined, the P4CONFIG file. The settings can be checked and changed before the connection to the server is established with the P4.connect() method. After your script connects, it can send multiple commands to the Perforce server with the same P4 instance. After the script is finished, it should disconnect from the server by calling the P4.disconnect() method.

The following example illustrates the basic structure of a P4Python script. The example establishes a connection, issues a command, and tests for errors resulting from the command.

```python
from P4 import P4, P4Exception    # Import the module
p4 = P4()                        # Create the P4 instance
p4.port = "1666"                  # Set some environment variables
p4.user = "fred"
try:                             # Catch exceptions with try/except
    p4.connect()                   # Connect to the Perforce server
    info = p4.run( "info" )        # Run "p4 info" (returns a dict)
    for key in info[0]:            # and display all key-value pairs
        print key, '=', info[0][key]
    p4.run( "edit", "file.txt" )   # Run "p4 edit file.txt"
except P4Exception:              # Display errors
    for e in p4.errors:
        print e
p4.disconnect()                  # Disconnect from the server
```

This example creates a client workspace from a template and syncs it:
from P4 import P4, P4Exception

template = "my-client-template"
client_root = "C:\work\my-root"
p4 = P4()

try:
    p4.connect()
    # Convert client spec into a Python dictionary
    client = p4.fetch_client( "-t", template )
    client._root = client_root
    p4.save_client( client )
    p4.run_sync()
except P4Exception:
    # If any errors occur, we'll jump in here. Just log them
    # and raise the exception up to the higher level

## Submitting a Changelist

This example creates a changelist, modifies it and then submits it:

from P4 import P4

p4 = P4()
p4.connect()
change = p4.fetch_change()

# Files were opened elsewhere and we want to
# submit a subset that we already know about.

myfiles = ["//depot/some/path/file1.c", "//depot/some/path/file1.h"]
change._description = "My changelist\nSubmitted from P4Python\n"
change._files = myfiles  # This attribute takes a Python list
p4.run_submit( change )

## Logging into Perforce using ticket-based authentication

On some servers, users might need to log in to Perforce before issuing commands. The following example illustrates login using Perforce tickets.
from P4 import P4

p4 = P4()
p4.user = "bruno"
p4.password = "my_password"
p4.connect()
p4.run_login()
opened = p4.run_opened()

...

### Connecting to Perforce over SSL

Scripts written with P4Python use any existing **P4TRUST** file present in their operating environment (by default, `.p4trust` in the home directory of the user that runs the script).

If the fingerprint returned by the server fails to match the one installed in the **P4TRUST** file associated with the script's run-time environment, your script will (and should!) fail to connect to the server.

### Changing your password

You can use P4Python to change your password, as shown in the following example:

from P4 import P4

p4 = P4()
p4.user = "bruno"
p4.password = "MyOldPassword"
p4.connect()

p4.run_password( "MyOldPassword", MyNewPassword" )

# p4.password is automatically updated with the encoded password

### Timestamp conversion

Timestamp information in P4Python is normally represented as seconds since Epoch (with the exception of **P4.Revision**). To convert this data to a more useful format, use the following procedure:

import datetime

...

myDate = datetime.datetime.utcfromtimestamp( int( timestampValue ) )
**Working with comments in specs**

As of P4Python 2012.3, comments in specs are preserved in the `parse_<spectype>()` and `format_<spectype>()` methods. This behavior can be circumvented by using `parse_spec( '<spectype>', spec )` and `format_spec( '<spectype>', spec )` instead of `parse_<spectype>( spec )` and `format_<spectype>( spec )`. For example:

```python
p4 = P4()
p4.connect()

# fetch a client spec in raw format, no formatting:
specform = p4.run( 'client', '-o', tagged=False )[0]

# convert the raw document into a spec
client1 = p4.parse_client( specform )

# comments are preserved in the spec as well
print( client1.comment )

# comments can be updated
client1.comment += '# ... and now for something completely different'

# the comment is prepended to the spec ready to be sent to the user
formatted1 = p4.format_client( client1 )

# or you can strip the comments
client2 = p4.parse_spec( 'client', specform )
formatted2 = p4.format_spec( 'client', specform )
```

**P4Python Classes**

The `P4` module consists of several public classes:

- “P4” on page 90
- “P4.P4Exception” on page 93
- “P4.DepotFile” on page 93
- “P4.Revision” on page 94
- “P4.Integration” on page 94
- “P4.Map” on page 94
- “P4.MergeData” on page 95
- “P4.Message” on page 96
- “P4.OutputHandler” on page 96
- “P4.Progress” on page 96
The following tables provide more details about each public class.

**P4**

Perforce client class. Handles connection and interaction with the Perforce server. There is one instance of each connection.

The following table lists attributes of the class `P4` in P4Python. The attributes are readable and writable unless indicated otherwise. The attributes can be strings, objects, or integers.

<table>
<thead>
<tr>
<th><strong>Attribute</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>api_level</code></td>
<td>API compatibility level. (Lock server output to a specified server level.)</td>
</tr>
<tr>
<td><code>charset</code></td>
<td>Charset for Unicode servers.</td>
</tr>
<tr>
<td><code>client</code></td>
<td><code>P4CLIENT</code>, the name of the client workspace to use.</td>
</tr>
<tr>
<td><code>cwd</code></td>
<td>Current working directory.</td>
</tr>
<tr>
<td><code>disable_tmp_cleanup</code></td>
<td>Disable cleanup of temporary objects.</td>
</tr>
<tr>
<td><code>encoding</code></td>
<td>Encoding to use when receiving strings from a non-Unicode server. If unset, use UTF8. Can be set to a legal Python encoding, or to raw to receive Python bytes instead of Unicode strings. Requires Python 3.</td>
</tr>
<tr>
<td><code>errors</code></td>
<td>An array containing the error messages received during execution of the last command.</td>
</tr>
<tr>
<td><code>exception_level</code></td>
<td>The exception level of the <code>P4</code> instance. Values can be:</td>
</tr>
<tr>
<td></td>
<td>• 0: no exceptions are raised.</td>
</tr>
<tr>
<td></td>
<td>• 1: only errors are raised as exceptions.</td>
</tr>
<tr>
<td></td>
<td>• 2: warnings are also raised as exceptions.</td>
</tr>
<tr>
<td></td>
<td>The default value is 2.</td>
</tr>
<tr>
<td><code>handler</code></td>
<td>An output handler.</td>
</tr>
<tr>
<td><code>host</code></td>
<td><code>P4HOST</code>, the name of the host used.</td>
</tr>
<tr>
<td><code>ignore_file</code></td>
<td>The path of the ignore file, <code>P4IGNORE</code>.</td>
</tr>
<tr>
<td><code>input</code></td>
<td>Input for the next command. Can be a string, a list or a dictionary.</td>
</tr>
<tr>
<td><code>maxlocktime</code></td>
<td>MaxLockTime used for all following commands</td>
</tr>
<tr>
<td><code>maxresults</code></td>
<td>MaxResults used for all following commands</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>maxscanrows</td>
<td>MaxScanRows used for all following commands.</td>
</tr>
<tr>
<td>messages</td>
<td>An array of <code>P4.Message</code> objects, one for each message sent by the server.</td>
</tr>
<tr>
<td>p4config_file</td>
<td>The location of the configuration file used (<code>P4CONFIG</code>). This attribute is read-only.</td>
</tr>
<tr>
<td>password</td>
<td><code>P4PASSWD</code>, the password used.</td>
</tr>
<tr>
<td>port</td>
<td><code>P4PORT</code>, the port used for the connection.</td>
</tr>
<tr>
<td>prog</td>
<td>The name of the script.</td>
</tr>
<tr>
<td>progress</td>
<td>A progress indicator.</td>
</tr>
<tr>
<td>server_case_insensitive</td>
<td>Detect whether or not the server is case sensitive.</td>
</tr>
<tr>
<td>server_level</td>
<td>Returns the current Perforce server level.</td>
</tr>
<tr>
<td>server_unicode</td>
<td>Detect whether or not the server is in Unicode mode.</td>
</tr>
<tr>
<td>streams</td>
<td>To disable streams support, set the value to 0 or <code>False</code>. By default, streams output is enabled for servers at 2011.1 or higher.</td>
</tr>
<tr>
<td>tagged</td>
<td>To disable tagged output for the following commands, set the value to 0 or <code>False</code>. By default, tagged output is enabled.</td>
</tr>
<tr>
<td>track</td>
<td>To enable performance tracking for the current connection, set the value to 1 or <code>True</code>. By default, server tracking is disabled.</td>
</tr>
<tr>
<td>track_output</td>
<td>If performance tracking is enabled, returns an array containing performance tracking information received during execution of the last command.</td>
</tr>
<tr>
<td>ticket_file</td>
<td><code>P4TICKETS</code>, the ticket file location used.</td>
</tr>
<tr>
<td>user</td>
<td><code>P4USER</code>, the user under which the connection is run.</td>
</tr>
<tr>
<td>version</td>
<td>The version of the script.</td>
</tr>
<tr>
<td>warnings</td>
<td>An array containing the warning messages received during execution of the last command.</td>
</tr>
</tbody>
</table>

The following table lists all public methods of the class `P4`. Many methods are wrappers around `P4.run()`, which sends a command to the Perforce server. Such methods are provided for your convenience.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>at_exception_level()</code></td>
<td>In the context of a <code>with</code> statement, temporarily set the exception level for the duration of a block.</td>
</tr>
<tr>
<td><code>clone()</code></td>
<td>Clones from another Perforce service into a local Perforce service, and returns a new P4 object.</td>
</tr>
<tr>
<td><code>connect()</code></td>
<td>Connects to the Perforce server.</td>
</tr>
<tr>
<td><code>connected()</code></td>
<td>Returns <code>True</code> if connected and the connection is alive, otherwise <code>False</code>.</td>
</tr>
<tr>
<td><code>delete &lt;spectype&gt;()</code></td>
<td>Deletes the spec <code>&lt;spectype&gt;</code>. Equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>P4.run( &quot;&lt;spectype&gt;&quot;, &quot;-d&quot; )</code></td>
</tr>
<tr>
<td><code>disconnect()</code></td>
<td>Disconnects from the Perforce server.</td>
</tr>
<tr>
<td><code>env()</code></td>
<td>Get the value of a Perforce environment variable, taking into account P4CONFIG files and (on Windows or OS X) the registry or user preferences.</td>
</tr>
<tr>
<td><code>fetch &lt;spectype&gt;()</code></td>
<td>Fetches the spec <code>&lt;spectype&gt;</code>. Equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>P4.run( &quot;&lt;spectype&gt;&quot;, &quot;-o&quot; )</code></td>
</tr>
<tr>
<td><code>format &lt;spectype&gt;()</code></td>
<td>Converts the spec <code>&lt;spectype&gt;</code> into a string.</td>
</tr>
<tr>
<td><code>identify()</code></td>
<td>Returns a string identifying the P4Python module.</td>
</tr>
<tr>
<td><code>init()</code></td>
<td>Initializes a new personal (local) Perforce server, and returns a new P4 object.</td>
</tr>
<tr>
<td><code>is_ignored()</code></td>
<td>Determines whether a particular file is ignored via the <code>P4IGNORE</code> feature.</td>
</tr>
<tr>
<td><code>iterate &lt;spectype&gt;()</code></td>
<td>Iterate through specs of form <code>&lt;spectype&gt;</code>.</td>
</tr>
<tr>
<td><code>P4()</code></td>
<td>Returns a new P4 object.</td>
</tr>
<tr>
<td><code>parse &lt;spectype&gt;()</code></td>
<td>Parses a string representation of the spec <code>&lt;spectype&gt;</code> and returns a dictionary.</td>
</tr>
<tr>
<td><code>run()</code></td>
<td>Runs a command on the server. Needs to be connected, or an exception is raised.</td>
</tr>
<tr>
<td><code>run_cmd()</code></td>
<td>Runs the command <code>cmd</code>. Equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>P4.run( &quot;command&quot; )</code></td>
</tr>
</tbody>
</table>
## Method | Description
---|---
run_filelog() | This command returns a list of `P4.DepotFile` objects. Specialization for the `P4.run()` method.
run_login() | Logs in using the specified password or ticket.
run_password() | Convenience method: updates the password. Takes two arguments: `oldpassword, newpassword`
run_resolve() | Interface to `p4 resolve`.
run_submit() | Convenience method for submitting changelists. When invoked with a change spec, it submits the spec. Equivalent to:

```python
p4.input = myspec.run( "submit", "-i" )
```
run_tickets() | Interface to `p4 tickets`.
save_<spectype>() | Saves the spec `<spectype>`. Equivalent to:

```python
P4.run( "<spectype>", "-i" )
```
set_env() | On Windows or OS X, set a variable in the registry or user preferences.
temp_client() | Creates a temporary client.
while_tagged() | In the context of a `with` statement, temporarily toggle tagged behavior for the duration of a block.

### P4.P4Exception

Exception class. Instances of this class are raised when errors and/or (depending on the `exception_level` setting) warnings are returned by the server. The exception contains the errors in the form of a string. `P4Exception` is a subclass of the standard Python `Exception` class.

### P4.DepotFile

Container class returned by `P4.run_filelog()`. Contains the name of the depot file and a list of `P4.Revision` objects.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
depotFile | Name of the depot file. |
revisions | List of `P4.Revision` objects |
Chapter 4. P4Python

P4.Revision

Container class containing one revision of a P4.DepotFile object.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Action that created the revision.</td>
</tr>
<tr>
<td>change</td>
<td>Changelist number</td>
</tr>
<tr>
<td>client</td>
<td>Client workspace used to create this revision.</td>
</tr>
<tr>
<td>desc</td>
<td>Short change list description.</td>
</tr>
<tr>
<td>depotFile</td>
<td>The name of the file in the depot.</td>
</tr>
<tr>
<td>digest</td>
<td>MD5 digest of the revision.</td>
</tr>
<tr>
<td>fileSize</td>
<td>File size of this revision.</td>
</tr>
<tr>
<td>integrations</td>
<td>List of P4.Integration objects.</td>
</tr>
<tr>
<td>rev</td>
<td>Revision.</td>
</tr>
<tr>
<td>time</td>
<td>Timestamp (as datetime.datetime object)</td>
</tr>
<tr>
<td>type</td>
<td>File type.</td>
</tr>
<tr>
<td>user</td>
<td>User that created this revision.</td>
</tr>
</tbody>
</table>

P4.Integration

Container class containing one integration for a P4.Revision object.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>Integration method (merge/branch/copy/ignored).</td>
</tr>
<tr>
<td>file</td>
<td>Integrated file.</td>
</tr>
<tr>
<td>srev</td>
<td>Start revision.</td>
</tr>
<tr>
<td>erev</td>
<td>End revision.</td>
</tr>
</tbody>
</table>

P4.Map

A class that allows users to create and work with Perforce mappings without requiring a connection to the Perforce server.
### P4.Map()

- **Method**: `P4.Map()`  
  **Description**: Construct a new Map object (class method).

- **Method**: `join()`  
  **Description**: Joins two maps to create a third (class method).

- **Method**: `clear()`  
  **Description**: Empties a map.

- **Method**: `count()`  
  **Description**: Returns the number of entries in a map.

- **Method**: `is_empty()`  
  **Description**: Tests whether or not a map object is empty.

- **Method**: `insert()`  
  **Description**: Inserts an entry into the map.

- **Method**: `translate()`  
  **Description**: Translate a string through a map.

- **Method**: `includes()`  
  **Description**: Tests whether a path is mapped.

- **Method**: `reverse()`  
  **Description**: Returns a new mapping with the left and right sides reversed.

- **Method**: `lhs()`  
  **Description**: Returns the left side as an array.

- **Method**: `rhs()`  
  **Description**: Returns the right side as an array.

- **Method**: `as_array()`  
  **Description**: Returns the map as an array.

### P4.MergeData

Class encapsulating the context of an individual merge during execution of a `p4 resolve` command. Passed to `P4.run_resolve()`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>your_name</td>
<td>Returns the name of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td>their_name</td>
<td>Returns the name of &quot;their&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td>base_name</td>
<td>Returns the name of &quot;base&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td>your_path</td>
<td>Returns the path of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td>their_path</td>
<td>Returns the path of &quot;their&quot; file in the merge. (temporary file on workstation into which <code>their_name</code> has been loaded)</td>
</tr>
<tr>
<td>base_path</td>
<td>Returns the path of the base file in the merge. (temporary file on workstation into which <code>base_name</code> has been loaded)</td>
</tr>
<tr>
<td>result_path</td>
<td>Returns the path to the merge result. (temporary file on workstation into which the automatic merge performed by the server has been loaded)</td>
</tr>
<tr>
<td>merge_hint</td>
<td>Returns hint from server as to how user might best resolve merge.</td>
</tr>
</tbody>
</table>
The `P4.MergeData` class also has one method:

```python
run_merge()
```
If the environment variable `P4MERGE` is defined, run it and return a boolean based on the return value of that program.

### P4.Message

Class for handling error messages in Perforce.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>severity</code></td>
<td>Returns the severity of the message.</td>
</tr>
<tr>
<td><code>generic</code></td>
<td>Returns the generic class of the error.</td>
</tr>
<tr>
<td><code>msgid</code></td>
<td>Returns the unique ID of the error message.</td>
</tr>
</tbody>
</table>

### P4.OutputHandler

Handler class that provides access to streaming output from the server; set `P4.handler` to an instance of a subclass of `P4.OutputHandler` to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outputBinary</code></td>
<td>Process binary data.</td>
</tr>
<tr>
<td><code>outputInfo</code></td>
<td>Process tabular data.</td>
</tr>
<tr>
<td><code>outputMessage</code></td>
<td>Process information or errors.</td>
</tr>
<tr>
<td><code>outputStat</code></td>
<td>Process tagged output.</td>
</tr>
<tr>
<td><code>outputText</code></td>
<td>Process text data.</td>
</tr>
</tbody>
</table>

### P4.Progress

Handler class that provides access to progress indicators from the server; set `P4.progress` to an instance of a subclass of `P4.Progress` to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init()</code></td>
<td>Initialize progress indicator as designated type.</td>
</tr>
<tr>
<td><code>setTotal()</code></td>
<td>Total number of units (if known).</td>
</tr>
<tr>
<td><code>setDescription()</code></td>
<td>Description and type of units to be used for progress reporting.</td>
</tr>
<tr>
<td><code>update()</code></td>
<td>If non-zero, user has requested a cancellation of the operation.</td>
</tr>
</tbody>
</table>
### P4.Resolver

Class for handling resolves in Perforce.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolve()</td>
<td>Perform a resolve and return the resolve decision as a string.</td>
</tr>
</tbody>
</table>

### P4.Spec

Class allowing access to the fields in a Perforce specification form.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>fieldname</em></td>
<td>Value associated with the field named <code>fieldname</code>.</td>
</tr>
<tr>
<td>comments</td>
<td>Array containing comments in a spec object.</td>
</tr>
<tr>
<td>permitted_fields</td>
<td>Array containing the names of the fields that are valid for this spec object.</td>
</tr>
</tbody>
</table>
Class P4

Description

Main interface to the Python client API.

This module provides an object-oriented interface to the Perforce version management system. Data is returned in Python arrays and dictionaries (hashes) and input can also be supplied in these formats.

Each P4 object represents a connection to the Perforce server, and multiple commands may be executed (serially) over a single connection (which of itself can result in substantially improved performance if executing long sequences of Perforce commands).

1. Instantiate your P4 object.

2. Specify your Perforce client environment:
   • client
   • host
   • password
   • port
   • user

3. Set any options to control output or error handling:
   • exception_level

4. Connect to the Perforce service.

   The Perforce protocol is not designed to support multiple concurrent queries over the same connection. Multithreaded applications that use the C++ API or derived APIs (including P4Python) should ensure that a separate connection is used for each thread, or that only one thread may use a shared connection at a time.

5. Run your Perforce commands.

6. Disconnect from the Perforce service.

Instance Attributes

p4.api_level -> int

Contains the API compatibility level desired. This is useful when writing scripts using Perforce commands that do not yet support tagged output. In these cases, upgrading to a later server that supports tagged output for the commands in question can break your script. Using this method
allows you to lock your script to the output format of an older Perforce release and facilitate seamless upgrades. Must be called before calling `P4.connect()`.

```python
define_p4level():
    from P4 import P4
    p4 = P4()
    p4.api_level = 67 # Lock to 2010.1 format
    p4.connect()
    ...
    p4.disconnect()
```

For the API integer levels that correspond to each Perforce release, see:

http://kb.perforce.com/article/512

**p4.charset -> string**

Contains the character set to use when connect to a Unicode enabled server. Do not use when working with non-Unicode-enabled servers. By default, the character set is the value of the `P4CHARSET` environment variable. If the character set is invalid, this method raises a `P4Exception`.

```python
define_p4charset():
    from P4 import P4
    p4 = P4()
    p4.charset = "iso8859-1"
    p4.connect()
    p4.run_sync()
    p4.disconnect()
```

**p4.client -> string**

Contains the name of your client workspace. By default, this is the value of the `P4CLIENT` taken from any `P4CONFIG` file present, or from the environment according to the normal Perforce conventions.

**p4.cwd -> string**

Contains the current working directly. Can be called prior to executing any Perforce command. Sometimes necessary if your script executes a `chdir()` as part of its processing.

```python
define_p4cwd():
    from P4 import P4
    p4 = P4()
    p4.cwd = "/home/bruno"
```

**p4.disable_tmp_cleanup -> string**

Invoke this prior to connecting if you need to use multiple P4 connections in parallel in a multi-threaded Python application.

```python
define_p4disable_tmp_cleanup():
    from P4 import P4
    p4 = P4()
```
```python
from P4 import P4
p4 = P4()
p4.disable_tmp_cleanup()
p4.connect()
...
p4.disconnect()
```

### p4.encoding -> string

When decoding strings from a non-Unicode server, strings are assumed to be encoded in UTF8. To use another encoding, set `p4.encoding` to a legal Python encoding, or `raw` to receive Python bytes instead of a Unicode string. Available only when compiled with Python 3.

### p4.errors -> list (read-only)

Returns an array containing the error messages received during execution of the last command.

```python
from P4 import P4, P4Exception
p4 = P4()
try:
    p4.connect()
    p4.exception_level = 1
    # ignore "File(s) up-to-date"
    files = p4.run_sync()
except P4Exception:
    for e in p4.errors:
        print e
finally:
    p4.disconnect()
```

### p4.exception_level -> int

Configures the events which give rise to exceptions. The following three levels are supported:

- **0**: disables all exception handling and makes the interface completely procedural; you are responsible for checking the `p4.errors` and `p4.warnings` arrays.
- **1**: causes exceptions to be raised only when errors are encountered.
- **2**: causes exceptions to be raised for both errors and warnings. This is the default.

For example:
from P4 import P4
p4 = P4()
p4.exception_level = 1
p4.connect()    # P4Exception on failure
p4.run_sync()   # File(s) up-to-date is a warning - no exception raised
p4.disconnect()

**p4.handler -> handler**

Set the output handler to a subclass of `P4.OutputHandler`.

**p4.host -> string**

Contains the name of the current host. It defaults to the value of `P4HOST` taken from any `P4CONFIG` file present, or from the environment as per the usual Perforce convention. Must be called before connecting to the Perforce server.

```python
from P4 import P4
p4 = P4()
p4.host = "workstation123.perforce.com"
p4.connect()
...
p4.disconnect()
```

**p4.ignore_file -> string**

Contains the path of the ignore file. It defaults to the value of `P4IGNORE`. Set `P4.ignore_file` prior to calling `P4.is_ignored()`.

```python
from P4 import P4
p4 = P4()
p4.connect()
p4.ignore_file = "/home/bruno/workspace/.ignore"
p4.disconnect()
```

**p4.input -> string | dict | list**

Contains input for the next command.

Set this attribute prior to running a command that requires input from the user. When the command requests input, the specified data is supplied to the command. Typically, commands of the form `p4 cmd -i` are invoked using the `P4.save_<spectype>()` methods, which retrieve the value from `p4.input` internally; there is no need to set `p4.input` when using the `P4.save_<spectype>()` shortcuts.

You may pass a string, a hash, or (for commands that take multiple inputs from the user) an array of strings or hashes. If you pass an array, note that the first element of the array will be popped each time Perforce asks the user for input.
For example, the following code supplies a description for the default changelist and then submits it to the depot:

```python
from P4 import P4
p4 = P4()
p4.connect()
change = p4.run_change( "-o" )[0]
change[ "Description" ] = "Autosubmitted changelist"
p4.input = change
p4.run_submit( "-i" )
p4.disconnect()
```

### p4.iterate_<spectype>( arguments ) -> P4.Spec

The `iterate_<spectype>()` methods are shortcut methods that allow you to quickly iterate through clients, labels, branches, etc. Valid `<spectypes>` are `clients`, `labels`, `branches`, `changes`, `streams`, `jobs`, `users`, `groups`, `depots` and `servers`. Valid arguments are any arguments that would be valid for the corresponding `run_<spectype>()` command.

For example:

```python
for client in p4.iterate_clients():
    # do something with the client spec
```

is equivalent to:

```python
for c in p4.run_clients():
    client = p4.fetch_client( c['client'] )
```

### p4.maxlocktime -> int

Limit the amount of time (in milliseconds) spent during data scans to prevent the server from locking tables for too long. Commands that take longer than the limit will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxlocktime` for information on the commands that support this limit.

### p4.maxresults -> int

Limit the number of results Perforce permits for subsequent commands. Commands that produce more than this number of results will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxresults` for information on the commands that support this limit.

### p4.maxscanrows -> int

Limit the number of database records Perforce scans for subsequent commands. Commands that attempt to scan more than this number of records will be aborted. The limit remains in force until you
disable it by setting it to zero. See `p4 help maxscanrows` for information on the commands that support this limit.

**p4.messages -> list (read-only)**

Returns a list of `P4.Message` objects, one for each message (info, warning or error) sent by the server.

**p4.p4config_file -> string (read-only)**

Contains the name of the current `P4CONFIG` file, if any. This attribute cannot be set.

**p4.password -> string**

Contains your Perforce password or login ticket. If not used, takes the value of `P4PASSWD` from any `P4CONFIG` file in effect, or from the environment according to the normal Perforce conventions.

This password is also used if you later call `p4.run_login()` to log in using the 2003.2 and later ticket system. After running `p4.run_login()`, the attribute contains the ticket the allocated by the server.

```python
from P4 import P4
p4 = P4()
p4.password = "mypass"
p4.connect()
p4.run_login()
```

**p4.port -> string**

Contains the host and port of the Perforce server to which you want to connect. It defaults to the value of `P4PORT` in any `P4CONFIG` file in effect, and then to the value of `P4PORT` taken from the environment.

```python
from P4 import P4
p4 = P4()
p4.port = "localhost:1666"
p4.connect()
...
```

**p4.prog -> string**

Contains the name of the program, as reported to Perforce system administrators running `p4 monitor show -e`. The default is `unnamed p4-python script`.

```python
from P4 import P4
p4 = P4()
p4.prog = "sync-script"
puts( p4.prog )
p4.connect
...
```
**p4.progress -> progress**

Set the progress indicator to a subclass of `P4.Progress`.

**p4.server_case_insensitive -> boolean**

Detects whether or not the server is case-sensitive.

**p4.server_level -> int (read-only)**

Returns the current Perforce server level. Each iteration of the Perforce server is given a level number. As part of the initial communication this value is passed between the client application and the Perforce server. This value is used to determine the communication that the Perforce server will understand. All subsequent requests can therefore be tailored to meet the requirements of this server level.

This attribute is 0 before the first command is run, and is set automatically after the first communication with the server.

For the API integer levels that correspond to each Perforce release, see:

[http://kb.perforce.com/article/571](http://kb.perforce.com/article/571)

**p4.server_unicode -> boolean**

Detects whether or not the server is in Unicode mode.

**p4.streams -> int**

If 1 or True, `p4.streams` enables support for streams. By default, streams support is enabled at 2011.1 or higher (`api_level >= 70`). Raises a `P4Exception` if you attempt to enable streams on a pre-2011.1 server. You can enable or disable support for streams both before and after connecting to the server.

```python
from P4 import P4
p4 = P4()
p4.streams = False
print p4.streams
```

**p4.tagged -> int**

If 1 or True, `p4.tagged` enables tagged output. By default, tagged output is on.

```python
from P4 import P4
p4 = P4()
p4.tagged = False
print p4.tagged
```
**p4.ticket_file -> string**

Contains the location of the `P4TICKETS` file.

**p4.track -> boolean**

If set to 1 or True, `p4.track` indicates that server performance tracking is enabled for this connection. By default, performance tracking is disabled.

**p4.track_output -> list (read-only)**

If performance tracking is enabled with `p4.track`, returns an array containing the performance data received during execution of the last command.

```python
from P4 import P4
p4 = P4()
p4.track = 1
p4.run_info()
print p4.track_output
```

**p4.user -> string**

Contains the Perforce username. It defaults to the value of `P4USER` taken from any `P4CONFIG` file present, or from the environment as per the usual Perforce convention.

```python
from P4 import P4
p4 = P4()
p4.user = "bruno"
p4.connect()
...p4.disconnect()
```

**p4.version -> string**

Contains the version of the program, as reported to Perforce system administrators in the server log.

```python
from P4 import P4
p4 = P4()
p4.version = "123"
puts( p4.version )
p4.connect()
...p4.disconnect()
```

**p4.warnings -> list (read-only)**

Contains the array of warnings that arose during execution of the last command.

```python
...
from P4 import P4, P4Exception
p4 = P4()

try:
    p4.connect()
    p4.exception_level = 2  # File(s) up-to-date is a warning
    files = p4.run_sync()
except P4Exception, ex:
    for w in p4.warnings:
        print w
finally:
    p4.disconnect()

## Class Methods

### P4.P4()

Construct a new P4 object. For example:

```
from P4 import P4
P4.P4()
```

### P4.clone( arguments... )

Clone from another Perforce service into a local Perforce server, and returns a new P4 object.

P4.clone() requires specification of the port of the source Perforce service from which files and version history should be cloned from, and either a remotespec or a filespec that specify which files and history to clone. For example, to clone using a remotespec:

```
import P4
p4 = P4.clone( "-p", "port", "-r", "remotespec" )
```

or to clone using a filespec:

```
import P4
p4 = P4.clone( "-p", "port", "-f", "filespec" )
```

The cloned instance inherits the case sensitivity and Unicode settings from the source Perforce service.

**Note**

All of the additional DVCS commands, such as p4 push or p4 switch, are available automatically in the usual fashion. For example: p4.run_push(). See “p4.run_<cmd>()” on page 113 for details.
**P4.identify()**

Return the version of P4Python that you are using.

```python
python -c "from P4 import P4; print P4.identify()"
```

The read-only string attributes `PATCHLEVEL` and `OS` are also available to test an installation of P4Python without having to parse the output of `P4.identify()`.

If applicable, `P4.identify()` also reports the version of the OpenSSL library used for building the underlying Perforce C++ API with which P4Python was built.

**P4.init([arguments])**

Initializes a new, personal (local) Perforce server, and returns a new `P4` object.

Without any arguments, `P4.init()` creates a new DVCS server in the current working directory, using the settings for case sensitivity and Unicode support from current environment variables.

`P4.init()` accepts the following keyword arguments:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>client</code></td>
<td>Workspace and server name</td>
<td><code>client=&quot;sknop-dvcs&quot;</code></td>
</tr>
<tr>
<td><code>user</code></td>
<td>Perforce username used for pushing</td>
<td><code>user=&quot;sven_erik_knop&quot;</code></td>
</tr>
<tr>
<td><code>directory</code></td>
<td>local path of the root directory for the new server</td>
<td><code>directory=&quot;/tmp/test-dvcs&quot;</code></td>
</tr>
<tr>
<td><code>casesensitive</code></td>
<td>specify case sensitivity</td>
<td><code>casesensitive=False</code></td>
</tr>
<tr>
<td><code>unicode</code></td>
<td>specify whether Unicode is enabled</td>
<td><code>unicode=True</code></td>
</tr>
</tbody>
</table>

```python
import P4
p4 = P4.init( directory="/Users/sknop/dvcs/" )
p4.connect()
# ...
p4.disconnect()
```

The `P4` instance returned by `P4.init()` has the port, user, and client workspace already set; all that is required for you is to connect to the server to perform any commands. Connection is not automatic, to give you an opportunity to set any protocol parameters; these parameters can only be set once before a connection is established.

**Note**

All of the additional DVCS commands, such as `p4 push` or `p4 switch`, are available automatically in the usual fashion. For example: `p4.run_push()`. See “p4.run_<cmd>()” on page 113 for details.
Instance Methods

p4.at_exception_level()

In the context of a `with` statement, temporarily set the exception level for a block. For example:

```python
from P4 import P4
p4 = P4()
p4.connect()
with p4.at_exception_level( P4.RAISE_ERRORS ):
    # no exceptions for warnings
    p4.run_sync( "//depot/main/..." )

    # exceptions back to normal...
```

p4.connect()

Initializes the Perforce client and connects to the server.

If the connection is successfully established, returns `None`. If the connection fails and `P4.exception_level` is 0, returns `False`, otherwise raises a `P4Exception`. If already connected, prints a message.

```python
from P4 import P4
p4 = P4()
p4.connect()
...
p4.disconnect()
```

P4.connect() returns a context management object that is usable with a `with` statement within a block; after the block is finished, the connection is automatically disconnected:

```python
import P4
p4 = P4.P4()
with p4.connect():
    # block in context of connection
    ...

    # p4 is disconnected outside the block
    ...
```

p4.connected() -> boolean

Returns `true` if connected to the Perforce server and the connection is alive, otherwise `false`. 
from P4 import P4
p4 = P4()

print p4.connected()
p4.connect()
print p4.connected()

```
p4.delete_<spectype>( [ options ], name ) -> list
```

The `delete_<spectype>()` methods are shortcut methods that allow you to delete the definitions of clients, labels, branches, etc. These methods are equivalent to:

```
p4.run( "<spectype>", '-d', [options], "spec name" )
```

The following code uses `P4.delete_client()` to delete client workspaces that have not been accessed in more than 365 days:

```
from P4 import P4, P4Exception
from datetime import datetime, timedelta

now = datetime.now()
p4 = P4()

try:
p4.connect()
for client in p4.run_clients():
    atime = datetime.fromtimestamp( int( client[ "Access" ] ) )
    # If the client has not been accessed for a year, delete it
    if ( atime + timedelta( 365 ) ) < now :
        p4.delete_client( '-f', client[ "client" ] )

except P4Exception:
    for e in p4.errors:
        print e

finally:
p4.disconnect()
```

```
p4.disconnect()
```

Disconnect from the Perforce server. Call this method before exiting your script.

```
from P4 import P4
p4 = P4()
p4.connect()
...
p4.disconnect()
```
p4.env(var)

Get the value of a Perforce environment variable, taking into account P4CONFIG files and (on Windows or OS X) the registry or user preferences.

```python
from P4 import P4
p4 = P4()
print p4.env( "P4PORT" )
```

p4.fetch_<spectype>() -> P4.Spec

The fetch_<spectype>() methods are shortcuts for running p4.run( "<spectype>", "-o" ).pop( 0 ). For example:

```python
label      = p4.fetch_label( "labelname" )
change     = p4.fetch_change( changeno )
clientspec = p4.fetch_client( "clientname" )
```

are equivalent to:

```python
label      = p4.run( "label", "-o", "labelname" )[0]
change     = p4.run( "change", "-o", changeno )[0]
clientspec = p4.run( "client", "-o", "clientname" )[0]
```

p4.format_spec("<spectype>", dict) -> string

Converts the fields in the dict containing the elements of a Perforce form (spec) into the string representation familiar to users. The first argument is the type of spec to format: for example, client, branch, label, and so on. The second argument is the hash to parse. There are shortcuts available for this method. You can use p4.format_<spectype>( dict ) instead of p4.format_spec( "<spectype>", dict ), where <spectype> is the name of a Perforce spec, such as client, label, etc.

p4.format_<spectype>( dict ) -> string

The format_<spectype>() methods are shortcut methods that allow you to quickly fetch the definitions of clients, labels, branches, etc. They’re equivalent to:

```python
p4.format_spec( "<spectype>", dict )
```

p4.is_ignored("<path>") -> boolean

Returns true if the <path> is ignored via the P4IGNORE feature. The <path> can be a local relative or absolute path.
```python
from P4 import P4
p4 = P4()

p4.connect()
if ( p4.is_ignored( "/home/bruno/workspace/file.txt" )):
    print "Ignored."
else:
    print "Not ignored."

p4.disconnect()
```

**p4.parse_spec("<spectype>", string) -> P4.Spec**

Parses a Perforce form (spec) in text form into a Python dict using the spec definition obtained from the server. The first argument is the type of spec to parse: client, branch, label, and so on. The second argument is the string buffer to parse.

There are shortcuts available for this method. You can use:

```python
p4.parse_<spectype>( buf )
```

instead of:

```python
p4.parse_spec( "<spectype>", buf )
```

where `<spectype>` is one of client, branch, label, and so on.

**p4.parse_<spectype>( string ) -> P4.Spec**

This is equivalent to:

```python
p4.parse_spec( "<spectype>", string )
```

For example, `parse_job( myJob )` converts the String representation of a job spec into a Spec object.

To parse a spec, P4 needs to have the spec available. When not connected to the Perforce server, P4 assumes the default format for the spec, which is hardcoded. This assumption can fail for jobs if the server's jobspec has been modified. In this case, your script can load a job from the server first with the command `p4.fetch_job( 'sometagname' )`, and P4 will cache and use the spec format in subsequent `p4.parse_job()` calls.

**p4.run("<cmd>",[arg,...])**

Base interface to all the run methods in this API. Runs the specified Perforce command with the arguments supplied. Arguments may be in any form as long as they can be converted to strings by `str()`.
The `p4.run()` method returns a list of results whether the command succeeds or fails; the list may, however, be empty. Whether the elements of the array are strings or dictionaries depends on:

i. server support for tagged output for the command, and

ii. whether tagged output was disabled by calling `p4.tagged = False`.

In the event of errors or warnings, and depending on the exception level in force at the time, `p4.run()` raises a `P4Exception`. If the current exception level is below the threshold for the error/warning, `p4.run()` returns the output as normal and the caller must explicitly review `p4.errors` and `p4.warnings` to check for errors or warnings.

```python
from P4 import P4
p4 = P4()
p4.connect()
spec = p4.run( "client", "-o" )[0]
p4.disconnect()
```

Shortcuts are available for `p4.run()`. For example:

```python
p4.run_command( args )
```

is equivalent to:

```python
p4.run( "command", args )
```

There are also some shortcuts for common commands such as editing Perforce forms and submitting. For example, this:

```python
from P4 import P4
p4 = P4()
p4.connect()
clientspec = p4.run_client( "-o" ).pop( 0 )
clientspec[ "Description" ] = "Build client"
p4.input = clientspec
p4.run_client( "-i" )
p4.disconnect()
```

...may be shortened to:

```python
from P4 import P4
p4 = P4()
p4.connect()
clientspec = p4.fetch_client()
clientspec[ "Description" ] = "Build client"
p4.save_client( clientspec )
p4.disconnect()
```
The following are equivalent:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>p4.delete_&lt;spectype&gt;()</td>
<td>p4.run( &quot;&lt;spectype&gt;&quot;, &quot;-d &quot; )</td>
</tr>
<tr>
<td>p4.fetch_&lt;spectype&gt;()</td>
<td>p4.run( &quot;&lt;spectype&gt;&quot;, &quot;-o &quot;).shift</td>
</tr>
<tr>
<td>p4.save_&lt;spectype&gt;( spec )</td>
<td>p4.input = spec</td>
</tr>
<tr>
<td></td>
<td>p4.run( &quot;&lt;spectype&gt;&quot;, &quot;-i&quot; )</td>
</tr>
</tbody>
</table>

As the commands associated with p4.fetch_<spectype>() typically return only one item, these methods do not return an array, but instead return the first result element.

For convenience in submitting changelists, changes returned by p4.fetch_change() can be passed to p4.run_submit(). For example:

```python
from P4 import P4
p4 = P4()
p4.connect()

spec = p4.fetch_change()
spec[ "Description" ] = "Automated change"
p4.run_submit( spec )
p4.disconnect()
```

**p4.run_<cmd>**

Shorthand for:

```python
p4.run( "<cmd>", arguments... )
```

**p4.run_filelog( <fileSpec> ) -> list**

Runs a p4 filelog on the fileSpec provided and returns an array of P4.DepotFile results (when executed in tagged mode), or an array of strings when executed in nontagged mode. By default, the raw output of p4 filelog is tagged; this method restructures the output into a more user-friendly (and object-oriented) form.

For example:
from P4 import P4, P4Exception
p4 = P4()

try:
    p4.connect()
    for r in p4.run_filelog( "index.html" )[0].revisions:
        for i in r.integrations:
            # Do something
except P4Exception:
    for e in p4.errors:
        print e
finally:
    p4.disconnect()

**p4.run_login(<arg>...) -> list**

Runs `p4 login` using a password or ticket set by the user.

**p4.run_password(oldpass, newpass) -> list**

A thin wrapper to make it easy to change your password. This method is (literally) equivalent to the following:

```python
p4.input( [ oldpass, newpass, newpass ] )
p4.run( "password" )
```

For example:

```python
from P4 import P4, P4Exception
p4 = P4()
p4.password = "myoldpass"

try:
    p4.connect()
    p4.run_password( "myoldpass", "mynewpass" )
except P4Exception:
    for e in p4.errors:
        print e
finally:
    p4.disconnect()
```

**p4.run_resolve([<resolver>], [arg...] -> list**

Run a `p4 resolve` command. Interactive resolves require the `<resolver>` parameter to be an object of a class derived from `P4.Resolver`. In these cases, the `P4.Resolver.resolve()` method is called to handle the resolve. For example:
p4.run_resolve ( resolver=MyResolver() )

To perform an automated merge that skips whenever conflicts are detected:

class MyResolver( P4.Resolver ):
    def resolve( self, mergeData ):
        if not mergeData.merge_hint == "e":
            return mergeData.merge_hint
        else:
            return "s" # skip the resolve, there is a conflict

In non-interactive resolves, no P4.Resolver object is required. For example:

p4.run_resolve ( "-at" )

p4.run_submit([ hash ], [ arg... ]) -> list

Submit a changelist to the server. To submit a changelist, set the fields of the changelist as required and supply any flags:

        change = p4.fetch_change()
        change._description = "Some description"
        p4.run_submit( "-r", change )

You can also submit a changelist by supplying the arguments as you would on the command line:

        p4.run_submit( "-d", "Some description", "somedir/..." )

p4.run_tickets() -> list

p4.run_tickets() returns an array of lists of the form (p4port, user, ticket) based on the contents of the local tickets file.

p4.save_<spectype>()>

The save_<spectype>() methods are shortcut methods that allow you to quickly update the definitions of clients, labels, branches, etc. They are equivalent to:

        p4.input = dictOrString
        p4.run( "<spectype>", "-i" )

For example:
from P4 import P4, P4Exception
p4 = P4()

try:
    p4.connect()
    client = p4.fetch_client()
    client[ "Owner" ] = p4.user
    p4.save_client( client )
except P4Exception:
    for e in p4.errors:
        print e

finally:
    p4.disconnect()

p4.set_env( var, value )

On Windows or OS X, set a variable in the registry or user preferences. To unset a variable, pass an empty string as the second argument. On other platforms, an exception is raised.

    p4.set_env = ( "P4CLIENT", "my_workspace" )
    p4.set_env = ( "P4CLIENT", "" )

p4.temp_client( "<prefix>" , "<template>" )

Creates a temporary client, using the prefix <prefix> and based upon a client template named <template>, then switches P4.client to the new client, and provides a temporary root directory. The prefix makes it easy to exclude the workspace from the spec depot.

This is intended to be used with a with statement within a block; after the block is finished, the temp client is automatically deleted and the temporary root is removed.

For example:

    from P4 import P4
    p4 = P4()
    p4.connect()
    with p4.temp_client( "temp", "my_template" ) as t:
        p4.run_sync()
        p4.run_edit( "foo" )
        p4.run_submit( "-dcomment" )

p4.while_tagged( boolean )

In the context of a with statement, enable or disable tagged behavior for the duration of a block. For example:
from P4 import P4
p4 = P4()
p4.connect()
with p4.while_tagged( False ):
    # tagged output disabled for this block
    print p4.run_info()

    # tagged output back to normal
...

Class P4.P4Exception

Description

Instances of this class are raised when P4 encounters an error or a warning from the server. The exception contains the errors in the form of a string. P4Exception is a shallow subclass of the standard Python Exception class.

Class Attributes

None.

Class Methods

None.
Class P4.DepotFile

Description

Utility class providing easy access to the attributes of a file in a Perforce depot. Each P4.DepotFile object contains summary information about the file and a list of revisions (P4.Revision objects) of that file. Currently, only the P4.run_filelog() method returns a list of P4.DepotFile objects.

Instance Attributes

df.depotFile -> string

Returns the name of the depot file to which this object refers.

df.revisions -> list

Returns a list of P4.Revision objects, one for each revision of the depot file.

Class Methods

None.

Instance Methods

None.
Class P4.Revision

Description

Utility class providing easy access to the revisions of P4.DepotFile objects. Created by P4.run_filelog().

Instance Attributes

rev.action -> string

Returns the name of the action which gave rise to this revision of the file.

rev.change -> int

Returns the change number that gave rise to this revision of the file.

rev.client -> string

Returns the name of the client from which this revision was submitted.

rev.depotFile -> string

Returns the name of the depot file to which this object refers.

rev.desc -> string

Returns the description of the change which created this revision. Note that only the first 31 characters are returned unless you use p4 filelog -L for the first 250 characters, or p4 filelog -l for the full text.

rev.digest -> string

Returns the MD5 digest of this revision.

rev.fileSize -> string

Returns this revision’s size in bytes.

rev.integrations -> list

Returns the list of P4.Integration objects for this revision.

rev.rev -> int

Returns the number of this revision of the file.
rev.time -> datetime
Returns the date/time that this revision was created.

rev.type -> string
Returns this revision's Perforce filetype.

rev.user -> string
Returns the name of the user who created this revision.

Class Methods
None.

Instance Methods
None.
Class P4.Integration

Description
Utility class providing easy access to the details of an integration record. Created by `P4.run_filelog()`.

Instance Attributes

integ.how -> string
Returns the type of the integration record - how that record was created.

integ.file -> string
Returns the path to the file being integrated to/from.

integ.srev -> int
Returns the start revision number used for this integration.

integ.erev -> int
Returns the end revision number used for this integration.

Class Methods
None.

Instance Methods
None.
Class P4.Map

Description

The P4.Map class allows users to create and work with Perforce mappings, without requiring a connection to a Perforce server.

Instance Attributes

None.

Class Methods

P4.Map([ list ]) -> P4.Map

Constructs a new P4.Map object.

P4.Map.join ( map1, map2 ) -> P4.Map

Join two P4.Map objects and create a third.

The new map is composed of the left-hand side of the first mapping, as joined to the right-hand side of the second mapping. For example:

```
# Map depot syntax to client syntax
client_map = P4.Map()
client_map.insert( "//depot/main/...", "//client/..." )

# Map client syntax to local syntax
client_root = P4.Map()
client_root.insert( "//client/...", "/home/bruno/workspace/..." )

# Join the previous mappings to map depot syntax to local syntax
local_map = P4.Map.join( client_map, client_root )
local_path = local_map.translate( "//depot/main/www/index.html" )

# local_path is now /home/bruno/workspace/www/index.html
```

Instance Methods

map.clear()

Empty a map.

map.count() -> int

Return the number of entries in a map.
map.is_empty() -> boolean

Test whether a map object is empty.

map.insert(string ...)  

Inserts an entry into the map.

May be called with one or two arguments. If called with one argument, the string is assumed to be a string containing either a half-map, or a string containing both halves of the mapping. In this form, mappings with embedded spaces must be quoted. If called with two arguments, each argument is assumed to be half of the mapping, and quotes are optional.

```python
# called with two arguments:
map.insert( "//depot/main/...", "//client/..." )

# called with one argument containing both halves of the mapping:
map.insert( "//depot/live/...//client/live/..." )

# called with one argument containing a half-map:
# This call produces the mapping "depot/... depot/..."
map.insert( "depot/..." )
```

map.translate(string, [boolean]) -> string

Translate a string through a map, and return the result. If the optional second argument is 1, translate forward, and if it is 0, translate in the reverse direction. By default, translation is in the forward direction.

map.includes(string) -> boolean

Tests whether a path is mapped or not.

```python
if map.includes( "//depot/main/..." ):
  ...
```

map.reverse() -> P4.Map

Return a new P4.Map object with the left and right sides of the mapping swapped. The original object is unchanged.

map.lhs() -> list

Returns the left side of a mapping as an array.

map.rhs() -> list

Returns the right side of a mapping as an array.
map.as_array() -> list

Returns the map as an array.
Class P4.MergeData

Description
Class containing the context for an individual merge during execution of a p4 resolve.

Instance Attributes

md.your_name -> string
Returns the name of "your" file in the merge. This is typically a path to a file in the workspace.

md.their_name -> string
Returns the name of "their" file in the merge. This is typically a path to a file in the depot.

md.base_name -> string
Returns the name of the "base" file in the merge. This is typically a path to a file in the depot.

md.your_path -> string
Returns the path of "your" file in the merge. This is typically a path to a file in the workspace.

md.their_path -> string
Returns the path of "their" file in the merge. This is typically a path to a temporary file on your local machine in which the contents of their_name have been loaded.

md.base_path -> string
Returns the path of the base file in the merge. This is typically a path to a temporary file on your local machine in which the contents of base_name have been loaded.

md.result_path -> string
Returns the path to the merge result. This is typically a path to a temporary file on your local machine in which the contents of the automatic merge performed by the server have been loaded.

md.merge_hint -> string
Returns the hint from the server as to how it thinks you might best resolve this merge.

Instance Methods

md.run_merge() -> boolean
If the environment variable P4MERGE is defined, md.run_merge() invokes the specified program and returns a boolean based on the return value of that program.
Class P4.Message

Description

P4.Message objects contain error or other diagnostic messages from the Perforce server; they are returned in P4.messages.

Script writers can test the severity of the messages in order to determine if the server message consisted of command output (E_INFO), warnings, (E_WARN), or errors (E_FAILED / E_FATAL).

Class Methods

None.

Instance Attributes

message.severity -> int

Severity of the message, which is one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_EMPTY</td>
<td>No error.</td>
</tr>
<tr>
<td>E_INFO</td>
<td>Informational message only.</td>
</tr>
<tr>
<td>E_WARN</td>
<td>Warning message only.</td>
</tr>
<tr>
<td>E_FAILED</td>
<td>Command failed.</td>
</tr>
<tr>
<td>E_FATAL</td>
<td>Severe error; cannot continue.</td>
</tr>
</tbody>
</table>

message.generic -> int

Returns the generic class of the error.

message.msgid -> int

Returns the unique ID of the message.
Class P4.OutputHandler

Description

The P4.OutputHandler class is a handler class that provides access to streaming output from the server. After defining the output handler, set p4.handler to an instance of a subclass of P4.OutputHandler, use p4.using_handler( MyHandler() ), or pass the handler as a named parameter for one statement only.

By default, P4.OutputHandler returns REPORT for all output methods. The different return options are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT</td>
<td>Messages added to output (don’t handle, don’t cancel)</td>
</tr>
<tr>
<td>HANDLED</td>
<td>Output is handled by class (don’t add message to output).</td>
</tr>
<tr>
<td>REPORT</td>
<td>CANCEL</td>
</tr>
<tr>
<td>HANDLED</td>
<td>CANCEL</td>
</tr>
</tbody>
</table>

Class Methods

class MyHandler(P4.OutputHandler)

Constructs a new subclass of P4.OutputHandler.

Instance Methods

outputBinary -> int

Process binary data.

outputInfo -> int

Process tabular data.

outputMessage -> int

Process informational or error messages.

outputStat -> int

Process tagged data.

outputText -> int

Process text data.
Class P4.Progress

Description

The P4.Progress class is a handler class that provides access to progress indicators from the server. After defining the progress class, set P4.progress to an instance of a subclass of P4.Progress, use p4.using_progress( MyProgress() ), or pass the progress indicator as a named parameter for one statement only.

You must implement all five of the following methods: init(), setDescription(), update(), setTotal(), and done(), even if the implementation consists of trivially returning 0.

Instance Attributes

None.

Class Methods

class MyProgress( P4.Progress )

Constructs a new subclass of P4.Progress.

Instance Methods

progress.init() -> int

Initialize progress indicator.

progress.setDescription( string, int ) -> int

Description and type of units to be used for progress reporting.

progress.update() -> int

If non-zero, user has requested a cancellation of the operation.

progress.setTotal( <total> ) -> int

Total number of units expected (if known).

progress.done() -> int

If non-zero, operation has failed.
Class P4.Resolver

Description

P4.Resolver is a class for handling resolves in Perforce. It is intended to be subclassed, and for subclasses to override the resolve() method. When P4.run_resolve() is called with a P4.Resolver object, it calls the P4.Resolver.resolve() method of the object once for each scheduled resolve.

Instance Attributes

None.

Class Methods

None.

Instance Methods

resolver.resolve( self, mergeData ) --> string

Returns the resolve decision as a string. The standard Perforce resolve strings apply:

<table>
<thead>
<tr>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ay</td>
<td>Accept Yours.</td>
</tr>
<tr>
<td>at</td>
<td>Accept Theirs.</td>
</tr>
<tr>
<td>am</td>
<td>Accept Merge result.</td>
</tr>
<tr>
<td>ae</td>
<td>Accept Edited result.</td>
</tr>
<tr>
<td>s</td>
<td>Skip this merge.</td>
</tr>
<tr>
<td>q</td>
<td>Abort the merge.</td>
</tr>
</tbody>
</table>

By default, all automatic merges are accepted, and all merges with conflicts are skipped. The P4.Resolver.resolve() method is called with a single parameter, which is a reference to a P4.MergeData object.
Class P4.Spec

Description

Utility class providing easy access to the attributes of the fields in a Perforce form.

Only valid field names may be set in a P4.Spec object. Only the field name is validated, not the content. Attributes provide easy access to the fields.

Instance Attributes

spec._<fieldname> -> string

Contains the value associated with the field named <fieldname>.

spec.comment -> dict

Contains an array containing the comments associated with the spec object.

spec.permitted_fields -> dict

Contains an array containing the names of fields that are valid in this spec object. This does not imply that values for all of these fields are actually set in this object, merely that you may choose to set values for any of these fields if you want to.

Class Methods

P4.Spec.new( dict ) -> P4.Spec

Constructs a new P4.Spec object given an array of valid fieldnames.

Instance Methods

None.
Introduction

P4PHP, the PHP interface to the Perforce API, enables you to write PHP code that interacts with a Perforce server. P4PHP enables your PHP scripts to:

- Get Perforce data and forms in arrays.
- Edit Perforce forms by modifying arrays.
- Provide exception-based error handling and optionally ignore warnings.
- Issue multiple commands on a single connection (performs better than spawning single commands and parsing the results).

The most recent release of P4PHP is 2013.1.

System Requirements

P4PHP is supported on Windows, Linux, FreeBSD, and OS X.

To build P4PHP from source, your development machine must also have:

- PHP 5.3.x, 5.4.x, or 5.5.x.
- The 2014.1 Perforce C/C++ API for your target platform
- The same C++ compiler used to build the Perforce C++ API on your target platform.

(If you get "unresolved symbol" errors when building or running P4PHP, you probably used the wrong compiler or the wrong Perforce API build.)

Installing P4PHP

You can download P4PHP from the Perforce web site:

http://www.perforce.com/product/components/apis

You must build the interface from source, as described in the release notes packaged with P4PHP.

Programming with P4PHP

The following example illustrates the basic structure of a P4PHP script. The example establishes a connection, issues a command, and tests for errors resulting from the command.
<?php
    $p4 = new P4();
    $p4->port = "1666";
    $p4->user = "fred";
    $p4->client = "fred-ws";

    try {
        $p4->connect();
        $info = $p4->run( "info" );
        foreach ( $info[0] as $key => $val ) {
            print "$key = $val\n";
        }
        $p4->run( "edit", "file.txt" );
        $p4->disconnect();
    } catch ( P4_Exception $e ) {
        print $e->getMessage() . "\n";
        foreach ( $p4->errors as $error ) {
            print "Error: $error\n";
        }
    }
?>

This example creates a client workspace from a template and syncs it:

<?php
    $template = "my-client-template";
    $client_root = "/home/user/work/my-root";
    $p4 = new P4();

    try {
        $p4->connect();

        // Convert client spec into an array
        $client = $p4->fetch_client( "-t", $template );
        $client["Root"] = $client_root;
        $p4->save_client( $client );
        $p4->run_sync();
    } catch ( P4_Exception $e ) {
        // If any errors occur, we'll jump in here. Just log them
        // and raise the exception up to the higher level
    }
?>

### Submitting a Changelist

This example creates a changelist, modifies it, and then submits it:
<?php
$p4 = new P4();
$p4->connect();
$change = $p4->fetch_change();

// Files were opened elsewhere and we want to
// submit a subset that we already know about.
$myfiles = array(
    '//depot/some/path/file1.c',
    '//depot/some/path/file1.h'
);

$change['description'] = "My changelist
Submitted from P4PHP
";
$change['files'] = $myfiles;
$p4->run_submit($change);
?>

Logging into Perforce using ticket-based authentication

On some servers, users might need to log in to Perforce before issuing commands. The following example illustrates login using Perforce tickets.

<?php
$p4 = new P4();
$p4->user = "bruno";
$p4->connect();
$p4->run_login('my_password');

$opened = $p4->
    run_opened();
?>

Connecting to Perforce over SSL

Scripts written with P4PHP use any existing P4TRUST file present in their operating environment (by default, .p4trust in the home directory of the user that runs the script).

If the fingerprint returned by the server fails to match the one installed in the P4TRUST file associated with the script's run-time environment, your script will (and should!) fail to connect to the server.

Changing your password

You can use P4PHP to change your password, as shown in the following example:
<?php
$p4 = new P4();
$p4->user = "bruno";
$p4->password = "MyOldPassword";
$p4->connect();

$p4->run_password( "MyOldPassword", "MyNewPassword" );
// $p4->password is automatically updated with the encoded password
?>

P4PHP Classes

The P4 module consists of several public classes:

- “P4” on page 136
- “P4_Exception” on page 139
- “P4_DepotFile” on page 139
- “P4_Revision” on page 139
- “P4_Integration” on page 140
- “P4_Map” on page 140
- “P4_MergeData” on page 141
- “P4_OutputHandlerAbstract” on page 141
- “P4_Resolver” on page 142

The following tables provide more details about each public class.

P4

Perforce client class. Handles connection and interaction with the Perforce server. There is one instance of each connection.

The following table lists properties of the class P4 in P4PHP. The properties are readable and writable unless indicated otherwise. The properties can be strings, arrays, or integers.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>api_level</td>
<td>API compatibility level. (Lock server output to a specified server level.)</td>
</tr>
<tr>
<td>charset</td>
<td>Charset for Unicode servers.</td>
</tr>
<tr>
<td>client</td>
<td>P4CLIENT, the name of the client workspace to use.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>cwd</strong></td>
<td>Current working directory.</td>
</tr>
<tr>
<td><strong>errors</strong></td>
<td>A read-only array containing the error messages received during execution of the last command.</td>
</tr>
<tr>
<td><strong>exception_level</strong></td>
<td>The exception level of the P4 instance. Values can be:</td>
</tr>
<tr>
<td></td>
<td>• 0: no exceptions are raised</td>
</tr>
<tr>
<td></td>
<td>• 1: only errors are raised as exceptions</td>
</tr>
<tr>
<td></td>
<td>• 2: warnings are also raised as exceptions</td>
</tr>
<tr>
<td></td>
<td>The default value is 2.</td>
</tr>
<tr>
<td><strong>expand_sequences</strong></td>
<td>Control whether keys with trailing numbers are expanded into arrays; by default, true, for backward-compatibility.</td>
</tr>
<tr>
<td><strong>handler</strong></td>
<td>An output handler.</td>
</tr>
<tr>
<td><strong>host</strong></td>
<td><strong>P4HOST</strong>, the name of the host used.</td>
</tr>
<tr>
<td><strong>input</strong></td>
<td>Input for the next command. Can be a string, or an array.</td>
</tr>
<tr>
<td><strong>maxlocktime</strong></td>
<td>MaxLockTime used for all following commands.</td>
</tr>
<tr>
<td><strong>maxresults</strong></td>
<td>MaxResults used for all following commands.</td>
</tr>
<tr>
<td><strong>maxscanrows</strong></td>
<td>MaxScanRows used for all following commands.</td>
</tr>
<tr>
<td><strong>p4config_file</strong></td>
<td>The location of the configuration file used (<strong>P4CONFIG</strong>). This property is read-only.</td>
</tr>
<tr>
<td><strong>password</strong></td>
<td><strong>P4PASSWD</strong>, the password used.</td>
</tr>
<tr>
<td><strong>port</strong></td>
<td><strong>P4PORT</strong>, the port used for the connection.</td>
</tr>
<tr>
<td><strong>prog</strong></td>
<td>The name of the script.</td>
</tr>
<tr>
<td><strong>server_level</strong></td>
<td>Returns the current Perforce server level. This property is read only.</td>
</tr>
<tr>
<td><strong>streams</strong></td>
<td>Enable or disable support for streams.</td>
</tr>
<tr>
<td><strong>tagged</strong></td>
<td>To disable tagged output for the following commands, set the value to 0 or False. By default, tagged output is enabled.</td>
</tr>
<tr>
<td><strong>ticket_file</strong></td>
<td><strong>P4TICKETS</strong>, the ticket file location used.</td>
</tr>
<tr>
<td><strong>user</strong></td>
<td><strong>P4USER</strong>, the user under which the connection is run.</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>The version of the script.</td>
</tr>
<tr>
<td>warnings</td>
<td>A read-only array containing the warning messages received during execution of the last command.</td>
</tr>
</tbody>
</table>

The following table lists all public methods of the class P4.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connect()</td>
<td>Connects to the Perforce server.</td>
</tr>
<tr>
<td>connected()</td>
<td>Returns <code>True</code> if connected and the connection is alive, otherwise <code>False</code>.</td>
</tr>
<tr>
<td>delete <code>&lt;spectype&gt;</code>()</td>
<td>Deletes the spec <code>&lt;spectype&gt;</code>. Equivalent to the command:</td>
</tr>
<tr>
<td></td>
<td>P4::run( &quot;&lt;spectype&gt;&quot;, &quot;-d&quot; );</td>
</tr>
<tr>
<td>disconnect()</td>
<td>Disconnects from the Perforce server.</td>
</tr>
<tr>
<td>env()</td>
<td>Get the value of a Perforce environment variable, taking into account P4CONFIG files and (on Windows or OS X) the registry or user preferences.</td>
</tr>
<tr>
<td>identify()</td>
<td>Returns a string identifying the P4PHP module. (This method is static.)</td>
</tr>
<tr>
<td>fetch <code>&lt;spectype&gt;</code>()</td>
<td>Fetches the spec <code>&lt;spectype&gt;</code>. Equivalent to the command:</td>
</tr>
<tr>
<td></td>
<td>P4::run( &quot;&lt;spectype&gt;&quot;, &quot;-o&quot; );</td>
</tr>
<tr>
<td>format <code>&lt;spectype&gt;</code>()</td>
<td>Converts the spec <code>&lt;spectype&gt;</code> into a string.</td>
</tr>
<tr>
<td>parse <code>&lt;spectype&gt;</code>()</td>
<td>Parses a string representation of the spec <code>&lt;spectype&gt;</code> and returns an array.</td>
</tr>
<tr>
<td>run()</td>
<td>Runs a command on the server. Needs to be connected, or an exception is raised.</td>
</tr>
<tr>
<td>run_cmd()</td>
<td>Runs the command <code>cmd</code>. Equivalent to:</td>
</tr>
<tr>
<td></td>
<td>P4::run( &quot;cmd&quot; );</td>
</tr>
<tr>
<td>run_filelog()</td>
<td>This command returns an array of P4_DepotFile objects. Specialization for the <code>run()</code> command.</td>
</tr>
<tr>
<td>run_login()</td>
<td>Logs in using the specified password or ticket.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>run_password()</code></td>
<td>Convenience method: updates the password. Takes two arguments: <code>oldpassword</code>, <code>newpassword</code>.</td>
</tr>
<tr>
<td><code>run_resolve()</code></td>
<td>Interface to <code>p4 resolve</code>.</td>
</tr>
<tr>
<td><code>run_submit()</code></td>
<td>Convenience method for submitting changelists. When invoked with a change spec, it submits the spec. Equivalent to:</td>
</tr>
<tr>
<td></td>
<td><code>p4::input = myspec; p4::run( &quot;submit&quot;, &quot;-i&quot; );</code></td>
</tr>
<tr>
<td><code>save_&lt;spectype&gt;()</code></td>
<td>Saves the spec <code>&lt;spectype&gt;</code>. Equivalent to the command:</td>
</tr>
<tr>
<td></td>
<td><code>P4::run( &quot;&lt;spectype&gt;&quot;, &quot;-i&quot; );</code></td>
</tr>
</tbody>
</table>

### P4_Exception

Exception class. Instances of this class are raised when errors and/or (depending on the `exception_level` setting) warnings are returned by the server. The exception contains the errors in the form of a string. `P4_Exception` extends the standard PHP `Exception` class.

### P4_DepotFile

Container class returned by `P4::run_filelog()`. Contains the name of the depot file and an array of `P4_Revision` objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>depotFile</code></td>
<td>Name of the depot file</td>
</tr>
<tr>
<td><code>revisions</code></td>
<td>Array of Revision objects.</td>
</tr>
</tbody>
</table>

### P4_Revision

Container class containing one revision of a `P4_DepotFile` object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>action</code></td>
<td>Action that created the revision.</td>
</tr>
<tr>
<td><code>change</code></td>
<td>Changelist number.</td>
</tr>
<tr>
<td><code>client</code></td>
<td>Client workspace used to create this revision.</td>
</tr>
</tbody>
</table>
Chapter 5. P4PHP

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Short changelist description.</td>
</tr>
<tr>
<td>depotFile</td>
<td>The name of the file in the depot.</td>
</tr>
<tr>
<td>digest</td>
<td>MD5 digest of the revision.</td>
</tr>
<tr>
<td>fileSize</td>
<td>File size of this revision.</td>
</tr>
<tr>
<td>integrations</td>
<td>Array of P4_Integration objects.</td>
</tr>
<tr>
<td>rev</td>
<td>Revision.</td>
</tr>
<tr>
<td>time</td>
<td>Timestamp.</td>
</tr>
<tr>
<td>type</td>
<td>File type.</td>
</tr>
<tr>
<td>user</td>
<td>User that created this revision.</td>
</tr>
</tbody>
</table>

**P4_Integration**

Container class containing one integration for a P4_Revision object.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>Integration method (merge/branch/copy/ignored).</td>
</tr>
<tr>
<td>file</td>
<td>Integrated file.</td>
</tr>
<tr>
<td>srev</td>
<td>Start revision.</td>
</tr>
<tr>
<td>erev</td>
<td>End revision.</td>
</tr>
</tbody>
</table>

**P4_Map**

A class that allows users to create and work with Perforce mappings without requiring a connection to the Perforce server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__construct()</td>
<td>Construct a new Map object.</td>
</tr>
<tr>
<td>join()</td>
<td>Joins two maps to create a third (static method).</td>
</tr>
<tr>
<td>clear()</td>
<td>Empties a map.</td>
</tr>
<tr>
<td>count()</td>
<td>Returns the number of entries in a map.</td>
</tr>
<tr>
<td>is_empty()</td>
<td>Tests whether or not a map object is empty.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>insert()</code></td>
<td>Inserts an entry into the map.</td>
</tr>
<tr>
<td><code>translate()</code></td>
<td>Translate a string through a map.</td>
</tr>
<tr>
<td><code>includes()</code></td>
<td>Tests whether a path is mapped.</td>
</tr>
<tr>
<td><code>reverse()</code></td>
<td>Returns a new mapping with the left and right sides reversed.</td>
</tr>
<tr>
<td><code>lhs()</code></td>
<td>Returns the left side as an array.</td>
</tr>
<tr>
<td><code>rhs()</code></td>
<td>Returns the right side as an array.</td>
</tr>
<tr>
<td><code>as_array()</code></td>
<td>Returns the map as an array.</td>
</tr>
</tbody>
</table>

**P4_MergeData**

Class encapsulating the context of an individual merge during execution of a `p4 resolve` command. Passed to `P4::run_resolve()`.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>your_name</code></td>
<td>Returns the name of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td><code>their_name</code></td>
<td>Returns the name of &quot;their&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td><code>base_name</code></td>
<td>Returns the name of &quot;base&quot; file in the merge. (file in the depot)</td>
</tr>
<tr>
<td><code>your_path</code></td>
<td>Returns the path of &quot;your&quot; file in the merge. (file in workspace)</td>
</tr>
<tr>
<td><code>their_path</code></td>
<td>Returns the path of &quot;their&quot; file in the merge. (temporary file on workstation into which <code>their_name</code> has been loaded)</td>
</tr>
<tr>
<td><code>base_path</code></td>
<td>Returns the path of the base file in the merge. (temporary file on workstation into which <code>base_name</code> has been loaded)</td>
</tr>
<tr>
<td><code>result_path</code></td>
<td>Returns the path to the merge result. (temporary file on workstation into which the automatic merge performed by the server has been loaded.)</td>
</tr>
<tr>
<td><code>merge_hint</code></td>
<td>Returns hint from server as to how user might best resolve merge.</td>
</tr>
</tbody>
</table>

**P4_OutputHandlerAbstract**

Handler class that provides access to streaming output from the server; set `$p4->handler` to an instance of a subclass of `P4_OutputHandlerAbstract` to enable callbacks:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outputBinary()</code></td>
<td>Process binary data.</td>
</tr>
</tbody>
</table>
## Chapter 5. P4PHP

### APIs for Scripting

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outputInfo()</code></td>
<td>Process tabular data.</td>
</tr>
<tr>
<td><code>outputMessage()</code></td>
<td>Process information or errors.</td>
</tr>
<tr>
<td><code>outputStat()</code></td>
<td>Process tagged output.</td>
</tr>
<tr>
<td><code>outputText()</code></td>
<td>Process text data.</td>
</tr>
</tbody>
</table>

### P4 Resolver

Abstract class for handling resolves in Perforce. This class must be subclassed in order to be used.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resolve()</code></td>
<td>Perform a resolve and return the resolve decision as a string.</td>
</tr>
</tbody>
</table>
Class P4

Description

Main interface to the PHP client API.

This module provides an object-oriented interface to the Perforce version management system. Data is returned in arrays and input can also be supplied in these formats.

Each P4 object represents a connection to the Perforce server, and multiple commands may be executed (serially) over a single connection (which of itself can result in substantially improved performance if executing long sequences of Perforce commands).

1. Instantiate your P4 object.

2. Specify your Perforce client environment:
   - client
   - host
   - password
   - port
   - user

3. Set any options to control output or error handling:
   - exception_level

4. Connect to the Perforce service.

   The Perforce protocol is not designed to support multiple concurrent queries over the same connection. Multithreaded applications that use the C++ API or derived APIs (including P4PHP) should ensure that a separate connection is used for each thread, or that only one thread may use a shared connection at a time.

5. Run your Perforce commands.

6. Disconnect from the Perforce service.

Properties

P4::api_level -> int

Contains the API compatibility level desired. This is useful when writing scripts using Perforce commands that do not yet support tagged output. In these cases, upgrading to a later server that supports tagged output for the commands in question can break your script. Using this method allows you to lock your script to the output format of an older Perforce release and facilitate seamless upgrades. Must be called before calling P4::connect().
<?php
$p4 = new P4();
$p4->api_level = 57; // Lock to 2005.1 format
$p4->connect();
...

$p4->disconnect();
?>

For the API integer levels that correspond to each Perforce release, see:

http://kb.perforce.com/article/512

**P4::charset -> string**

Contains the character set to use when connect to a Unicode enabled server. Do not use when working with non-Unicode-enabled servers. By default, the character set is the value of the P4CHARSET environment variable. If the character set is invalid, this method raises a P4_Exception.

<?php
$p4 = new P4();
$p4->client = "www";
$p4->charset = "iso8859-1";

$p4->connect();
$p4->run_sync();
$p4->disconnect();
?>

**P4::client -> string**

Contains the name of your client workspace. By default, this is the value of the P4CLIENT taken from any P4CONFIG file present, or from the environment according to the normal Perforce conventions.

**P4::cwd -> string**

Contains the current working directly. Can be called prior to executing any Perforce command. Sometimes necessary if your script executes a chdir() as part of its processing.

<?php
$p4 = new P4();
$p4->cwd = "/home/bruno"
?>
P4::errors -> array (read-only)

Returns an array containing the error messages received during execution of the last command.

<?php
$p4 = new P4();
$p4->connect();
$p4->exception_level = 1;
$p4->connect();  // P4_Exception on failure
$p4->run_sync(); // File(s) up-to-date is a warning; no exception raised

$err = $p4->errors;
print_r($err);

$p4->disconnect();
?>

P4::exception_level -> int

Configures the events which give rise to exceptions. The following three levels are supported:

- 0: disables all exception handling and makes the interface completely procedural; you are responsible for checking the P4::errors and P4::warnings arrays.
- 1: causes exceptions to be raised only when errors are encountered.
- 2: causes exceptions to be raised for both errors and warnings. This is the default.

For example:

<?php
$p4 = new P4();
$p4->exception_level = 1;
$p4->connect();  // P4_Exception on failure
$p4->run_sync(); // File(s) up-to-date is a warning; no exception raised
$p4->disconnect();
?>

P4::expand_sequences -> bool

Controls whether keys with trailing numbers are expanded into arrays when using tagged output. By default, expand_sequences is true to maintain backwards compatibility. Expansion can be enabled and disabled on a per-command basis.

For example:
<?php
$p4 = new P4();
$p4->connect();
$p4->expand_sequences = false; // disables sequence expansion.
$result = $p4->run('fstat', '-Oa', '//depot/path/...');
var_dump($result);
?>

**P4::handler -> handler**

Contains the output handler.

**P4::host -> string**

Contains the name of the current host. It defaults to the value of P4HOST taken from any P4CONFIG file present, or from the environment as per the usual Perforce convention. Must be called before connecting to the Perforce server.

<?php
$p4 = new P4();
$p4->host = "workstation123.perforce.com";
$p4->connect();
?>

**P4::input -> string | array**

Contains input for the next command.

Set this property prior to running a command that requires input from the user. When the command requests input, the specified data is supplied to the command. Typically, commands of the form `p4 cmd -i` are invoked using the `P4::save_<spectype>()` methods, which retrieve the value from `P4::input` internally; there is no need to set `P4::input` when using the `P4::save_<spectype>()` shortcuts.

You may pass a string, an array, or (for commands that take multiple inputs from the user) an array of strings or arrays. If you pass an array, note that the first element of the array will be popped each time Perforce asks the user for input.

For example, the following code supplies a description for the default changelist and then submits it to the depot:
<?php
$p4 = new P4();
$p4->connect();

$change = $p4->run_change("-o")[0];
$change[ 'Description' ] = "Autosubmitted changelist";
$p4->input = $change;
$p4->run_submit("-i");
$p4->disconnect();
?>

**P4::maxlocktime -> int**

Limit the amount of time (in milliseconds) spent during data scans to prevent the server from locking tables for too long. Commands that take longer than the limit will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxlocktime` for information on the commands that support this limit.

**P4::maxresults -> int**

Limit the number of results Perforce permits for subsequent commands. Commands that produce more than this number of results will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxresults` for information on the commands that support this limit.

**P4::maxscanrows -> int**

Limit the number of database records Perforce scans for subsequent commands. Commands that attempt to scan more than this number of records will be aborted. The limit remains in force until you disable it by setting it to zero. See `p4 help maxscanrows` for information on the commands that support this limit.

**P4::p4config_file -> string (read-only)**

Contains the name of the current P4CONFIG file, if any. This property cannot be set.

**P4::password -> string**

Contains your Perforce password or login ticket. If not used, takes the value of `P4PASSWD` from any P4CONFIG file in effect, or from the environment according to the normal Perforce conventions.

This password is also used if you later call `P4::run_login()` to log in using the 2003.2 and later ticket system. After running `P4::run_login()`, the property contains the ticket the allocated by the server.
<?php
$p4 = new P4();
$p4->password = "mypass";
$p4->connect();
$p4->run_login();
...
$p4->disconnect();
?>

**P4::port -> string**

Contains the host and port of the Perforce server to which you want to connect. It defaults to the value of `P4PORT` in any `P4CONFIG` file in effect, and then to the value of `P4PORT` taken from the environment.

```php
<?php
$p4 = new P4();
$p4->port = "localhost:1666";
$p4->connect();
...
$p4->disconnect();
?>
```

**P4::prog -> string**

Contains the name of the program, as reported to Perforce system administrators running `p4 monitor show -e`. The default is unnamed `p4-php script`.

```php
<?php
$p4 = new P4();
$p4->prog = "sync-script";
print $p4->prog;
$p4->connect();
...
$p4->disconnect();
?>
```
### P4::server_level -> int (read-only)

Returns the current Perforce server level. Each iteration of the Perforce server is given a level number. As part of the initial communication this value is passed between the client application and the Perforce server. This value is used to determine the communication that the Perforce server will understand. All subsequent requests can therefore be tailored to meet the requirements of this server level.

This property is 0 before the first command is run, and is set automatically after the first communication with the server.

For the API integer levels that correspond to each Perforce release, see:

[http://kb.perforce.com/article/571](http://kb.perforce.com/article/571)

### P4::streams -> bool

If true, P4::streams enables support for streams. By default, streams support is enabled at 2011.1 or higher (api_level >= 70). Raises a P4Exception if you attempt to enable streams on a pre-2011.1 server. You can enable or disable support for streams both before and after connecting to the server.

```php
<?php
$p4 = new P4();
$p4->streams = false;
print $p4->streams;
?>
```

### P4::tagged -> bool

If true, P4::tagged enables tagged output. By default, tagged output is on.

```php
<?php
tagged = false;
print $p4->tagged;
?>
```

### P4::ticket_file -> string

Contains the location of the P4TICKETS file.

### P4::user -> string

Contains the Perforce username. It defaults to the value of P4USER taken from any P4CONFIG file present, or from the environment as per the usual Perforce convention.
<?php
$p4 = new P4();
$p4->user = "bruno";
$p4->connect();
...
P4::disconnect();
?>

**P4::version -> string**

Contains the version of the program, as reported to Perforce system administrators in the server log.

```php
<?php
$p4 = new P4();
$p4->version = "123";
print $p4->version;
$p4->connect();
...
$p4->disconnect();
?>
```

**P4::warnings -> array (read-only)**

Contains the array of warnings that arose during execution of the last command.

```php
<?php
$p4 = new P4();
$p4->connect(); // P4_Exception on failure
$p4->exception_level = 2;

$files = $p4->run_sync();
$warn = $p4->warnings;
print_r($warn);

$p4->disconnect();
?>
```

**Constructor**

**P4::__construct**

Construct a new P4 object. For example:
<?php
    $p4 = new P4();
?>

Static Methods

P4::identify() \rightarrow string

Return the version of P4PHP that you are using, and, if applicable, the version of the OpenSSL library used for building the underlying Perforce C++ API with which P4PHP was built).

<?php
    print P4::identify();
?>

produces output similar to the following:

Perforce - The Fast Software Configuration Management System.
Copyright 1995-2013 Perforce Software. All rights reserved.

Instance Methods

P4::connect() \rightarrow bool

Initializes the Perforce client and connects to the server.

If the connection is successfully established, returns None. If the connection fails and exception_level is 0, returns False, otherwise raises a P4_Exception. If already connected, prints a message.

<?php
    $p4 = new P4();
    $p4->connect();
    ...
    $p4->disconnect();
?>

P4::connected() \rightarrow bool

Returns true if connected to the Perforce server and the connection is alive, otherwise false.
<?php
$p4 = new P4();
if ( !$p4->connected() ) {
    print "Not Connected\n";
}

$p4->connect();
if ( $p4->connected() ) {
    print "Connected\n";
}

$p4->disconnect();
?>

**P4::delete_<spectype>( [ options ], name ) -> array**

The `delete_<spectype>()` methods are shortcut methods that allow you to delete the definitions of clients, labels, branches, etc. These methods are equivalent to:

```php
P4::run( "<spectypes>", '-d', [options], "spec name" );
```

The following code uses `P4::delete_client()` to delete client workspaces that have not been accessed in more than 365 days:

```php
<?php
$p4 = new P4();
try {
    $p4->connect();
    foreach ( $p4->run_clients() as $client) {
        $atime = int( $client['Access'] );
        // If the client has not been accessed for a year, delete it
        if ( (time() - $atime) > 31536000 ) { // seconds in 365 days
            $p4->delete_client( "-f", $client["Client"] );
        }
    }
} catch ( P4_Exception $e ) {
    print $e->getMessage() . "\n";
    foreach ( $p4->errors as $error ) {
        print "Error: $error\n";
    }
}
?>
```

**P4::disconnect() -> void**

Disconnect from the Perforce server. Call this method before exiting your script.
<?php
$p4 = new P4();
$p4->connect();
...
$p4->disconnect();
?>

**P4::env( var )** -> string

Get the value of a Perforce environment variable, taking into account `P4CONFIG` files and (on Windows or OS X) the registry or user preferences.

```php
<?php
$p4 = new P4();
print $p4->env( "P4PORT" );
?>
```

**P4::fetch_<spectype>()** -> array

The `fetch_<spectype>()` methods are shortcuts for running `$p4->run( "<spectype>", "-o" )` and returning the first element of the array. For example:

```php
$label      = $p4->fetch_label( "labelname" );
$change     = $p4->fetch_change( changeno);
$clientspec = $p4->fetch_client( "clientname" );
```

are equivalent to:

```php
$label      = $p4->run( "label", "-o", "labelname" );
$change     = $p4->run( "change", "-o", changeno );
$clientspec = $p4->run( "client", "-o", clientname );
```

**P4::format_spec( "<spectype>", array )** -> string

Converts the fields in the array containing the elements of a Perforce form (spec) into the string representation familiar to users. The first argument is the type of spec to format: for example, client, branch, label, and so on. The second argument is the hash to parse. There are shortcuts available for this method. You can use `$p4->format_<spectype>( array )` instead of `$p4->format_spec( "<spectype>", array )`, where `<spectype>` is the name of a Perforce spec, such as client, label, etc.
**P4::format_<spectype>( array ) -> string**

The `format_<spectype>()` methods are shortcut methods that allow you to quickly fetch the definitions of clients, labels, branches, etc. They’re equivalent to:

```php
$p4->format_spec( "<spectype>", array );
```

**P4::parse_spec("<spectype>", string ) -> array**

Parses a Perforce form (spec) in text form into an array using the spec definition obtained from the server. The first argument is the type of spec to parse: client, branch, label, and so on. The second argument is the string buffer to parse.

There are shortcuts available for this method. You can use:

```php
$p4->parse_<spectype>( buf );
```

instead of:

```php
$p4->parse_spec( "<spectype>", buf );
```

where `<spectype>` is one of client, branch, label, and so on.

**P4::parse_<spectype>( string ) -> array**

This is equivalent to:

```php
$p4->parse_spec( "<spectype>", string )
```

For example:

```php
$p4->parse_job( myJob );
```

converts the String representation of a job spec into an array.

To parse a spec, P4 needs to have the spec available. When not connected to the Perforce server, P4 assumes the default format for the spec, which is hardcoded. This assumption can fail for jobs if the server’s jobspec has been modified. In this case, your script can load a job from the server first with the command `fetch_job( "somename")`, and P4 will cache and use the spec format in subsequent `P4::parse_job()` calls.

**P4::run( <cmd>, [arg, ...] ) -> mixed**

Base interface to all the run methods in this API. Runs the specified Perforce command with the arguments supplied. Arguments may be in any form as long as they can be converted to strings.
The `P4::run()` method returns an array of results whether the command succeeds or fails; the array may, however, be empty. Whether the elements of the array are strings or arrays depends on:

i. server support for tagged output for the command, and

ii. whether tagged output was disabled by calling `$p4->tagged = false`.

In the event of errors or warnings, and depending on the exception level in force at the time, `P4::run()` raises a `P4_Exception`. If the current exception level is below the threshold for the error/warning, `P4::run()` returns the output as normal and the caller must explicitly review `P4::errors` and `P4::warnings` to check for errors or warnings.

```php
<?php
$p4 = new P4();
print $p4->env("P4PORT");
$p4->connect();
$spec = $p4->run("client", "-o")[0];
$p4->disconnect();
?>
```

Shortcuts are available for `P4::run`. For example:

```php
$p4->run_command("args");
```

is equivalent to:

```php
$p4->run("command", args);
```

There are also some shortcuts for common commands such as editing Perforce forms and submitting. For example, this:

```php
<?php
$p4 = new P4();
$p4->connect();
$clientspec = array_pop($p4->run_client("-o"));
$clientspec["Description"] = "Build Client";
$p4->input = $clientspec;
$p4->run_client("-i");
$p4->disconnect();
?>
```
may be shortened to:

```php
<?php
 $p4 = new P4();
 $p4->connect();

 $clientspec = $p4->fetch_spec();
 $clientspec['Description'] = "Build client";

 $p4->save_client( $clientspec );
 $p4->disconnect();
 ?>
```

The following are equivalent:

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p4-&gt;delete_&lt;spectype&gt;()</td>
<td>$p4-&gt;run( &quot;&lt;spectype&gt;&quot;, &quot;-d &quot; );</td>
</tr>
<tr>
<td>$p4-&gt;fetch_&lt;spectype&gt;()</td>
<td>array_shift( $p4-&gt;run( &quot;&lt;spectype&gt;&quot;, &quot;-o &quot; ) );</td>
</tr>
</tbody>
</table>
| $p4->save_<spectype>( spec ) | $p4->input = $spec;

    $p4->run( "<spectype>", "-i" );

As the commands associated with P4::fetch_<spectype>() typically return only one item, these methods do not return an array, but instead return the first result element.

For convenience in submitting changelists, changes returned by P4::fetch_change() can be passed to P4::run_submit(). For example:

```php
<?php
 $p4 = new P4();
 $p4->connect();

 $spec = $p4->fetch_change();
 $spec['Description'] = "Automated change";
 $p4->run_submit( $spec );

 $p4->disconnect();
 ?>
```

**P4::run_<cmd>() -> mixed**

Shorthand for:
P4::run( "cmd", arguments... );

P4::run_filelog(<fileSpec>) -> array

Runs a p4 filelog on the fileSpec provided and returns an array of P4_DepotFile results (when executed in tagged mode), or an array of strings when executed in nontagged mode. By default, the raw output of p4 filelog is tagged; this method restructures the output into a more user-friendly (and object-oriented) form.

For example:

```php
<?php
$p4 = new P4();
try {
    $p4->connect();
    $filelog = $p4->run_filelog( "index.html" );
    foreach ( $filelog->revisions as $revision ) {
        // do something
    }
} catch ( P4_Exception $e ) {
    print $e->getMessage() . "\n";
    foreach ( $p4->errors as $error ) {
        print "Error: $error\n";
    }
}?
```

P4::run_login(arg...) -> array

Runs p4 login using a password or ticket set by the user.

P4::run_password(oldpass, newpass) -> array

A thin wrapper to make it easy to change your password. This method is equivalent to the following:

```php
<?php
$p4->input = array( $oldpass, $newpass, $newpass );
$p4->run( "password" );
?>
```

For example:
<?php
$p4 = new P4();
$p4->password = "myoldpass";

try {
    $p4->connect();
    $p4->run_password("myoldpass", "mynnewpass");
    $p4->disconnect();
} catch (P4_Exception $e) {
    print $e->getMessage() . "\n";
    foreach ($p4->errors as $error) {
        print "Error: $error\n";
    }
}
?>

P4::runresolve( [<resolver>], [arg...] ) -> array

Run a p4 resolve command. Interactive resolves require the <resolver> parameter to be an object of a class derived from P4_Resolver. In these cases, the P4::Resolver::resolve() method is called to handle the resolve. For example:

<?php
$p4->runresolve( new MyResolver() );
?>

To perform an automated merge that skips whenever conflicts are detected:

<?php
class MyResolver extends P4_Resolver {
    public function resolve( $merge_data ) {
        if ( $merge_data->merge_hint != 'e' ) {
            return $merge_data->merge_hint;
        } else {
            return "s"; // skip, there's a conflict
        }
    }
}
?>

In non-interactive resolves, no P4_Resolver object is required. For example:

$p4->runresolve( "-at" );
**P4::run_submit([array], [arg...]) -> array**

Submit a changelist to the server. To submit a changelist, set the fields of the changelist as required and supply any flags:

```php
$p4->change = $p4->fetch_change();
$change["Description"] = "Some description";
$p4->run_submit( "-r", $change );
```

You can also submit a changelist by supplying the arguments as you would on the command line:

```php
$p4->run_submit( "-d", "Some description", "somedir/..." );
```

**P4::save_<spectype>()**

The `save_<spectype>()` methods are shortcut methods that allow you to quickly update the definitions of clients, labels, branches, etc. They are equivalent to:

```php
$p4->input = $arrayOrString;
$p4->run( "<spectype>", "-i" );
```

For example:

```php
<?php
$p4 = new P4();
try {
    $p4->connect();
    $client = $p4->fetch_client();
    $client["Owner"] = $p4->user;
    $p4->save_client( $client );
    $p4->disconnect();
} catch ( P4_Exception $e ) {
    print $e->getMessage() . "\n";
    foreach ( $p4->errors as $error ) {
        print "Error: $error\n";
    }
}
?>
```
Class P4_Exception

**Description**

Instances of this class are raised when P4 encounters an error or a warning from the server. The exception contains the errors in the form of a string. P4_Exception is an extension of the standard Exception class.

**Class Attributes**

None.

**Static Methods**

None.
Class P4_DepotFile

Description

Utility class providing easy access to the attributes of a file in a Perforce depot. Each P4_DepotFile object contains summary information about the file and an array of revisions (P4_Revision objects) of that file. Currently, only the P4::run_filelog() method returns an array of P4_DepotFile objects.

Properties

$df->depotFile -> string

Returns the name of the depot file to which this object refers.

$df->revisions -> array

Returns an array of P4_Revision objects, one for each revision of the depot file.

Static Methods

None.

Instance Methods

None.
Class P4_Revision

Description

Utility class providing easy access to the revisions of P4_DepotFile objects. Created by P4::run_filelog().

Properties

$rev->action -> string

Returns the name of the action which gave rise to this revision of the file.

$rev->change -> long

Returns the change number that gave rise to this revision of the file.

$rev->client -> string

Returns the name of the client from which this revision was submitted.

$rev->depotFile -> string

Returns the name of the depot file to which this object refers.

$rev->desc -> string

Returns the description of the change which created this revision. Note that only the first 31 characters are returned unless you use p4 filelog -L for the first 250 characters, or p4 filelog -l for the full text.

$rev->digest -> string

Returns the MD5 digest of this revision.

$rev->fileSize -> long

Returns this revision’s size in bytes.

$rev->integrations -> array

Returns the array of P4_Integration objects for this revision.

$rev->rev -> long

Returns the number of this revision of the file.
Chapter 5. P4PHP

$rev->time -> string
Returns the date/time that this revision was created.

$rev->type -> string
Returns this revision's Perforce filetype.

$rev->user -> string
Returns the name of the user who created this revision.

**Static Methods**

None.

**Instance Methods**

None.
Class P4_Integration

**Description**
Utility class providing easy access to the details of an integration record. Created by P4::run_filelog().

**Properties**

$integ->how -> string
Returns the type of the integration record - how that record was created.

$integ->file -> string
Returns the path to the file being integrated to/from.

$integ->srev -> int
Returns the start revision number used for this integration.

$integ->erev -> int
Returns the end revision number used for this integration.

**Static Methods**
None.

**Instance Methods**
None.
Class P4_Map

Description
The P4_Map class allows users to create and work with Perforce mappings, without requiring a connection to a Perforce server.

Properties
None.

Constructor

P4_Map::__construct([ array ]) -> P4_Map

Constructs a new P4_Map object.

Static Methods

P4_Map::join ( map1, map2 ) -> P4_Map

Join two P4_Map objects and create a third P4_Map. The new map is composed of the left-hand side of the first mapping, as joined to the right-hand side of the second mapping. For example:

```php
// Map depot syntax to client syntax
$client_map = new P4_Map();
$client_map->insert( "//depot/main/...", "//client/..." );

// Map client syntax to local syntax
$client_root = new P4_Map();
$client_root->insert( "//client/...", "/home/bruno/workspace/..." );

// Join the previous mappings to map depot syntax to local syntax
$local_map = P4_Map::join( $client_map, $client_root );
$local_path = $local_map->translate( "//depot/main/www/index.html" );

// local_path is now /home/bruno/workspace/www/index.html
```

Instance Methods

$map->clear() -> void

Empty a map.

$map->count() -> int

Return the number of entries in a map.
$map->is_empty() -> bool

Test whether a map object is empty.

$map->insert( string ...) -> void

Inserts an entry into the map.

May be called with one or two arguments. If called with one argument, the string is assumed to be a string containing either a half-map, or a string containing both halves of the mapping. In this form, mappings with embedded spaces must be quoted. If called with two arguments, each argument is assumed to be half of the mapping, and quotes are optional.

```php
// called with two arguments:
$map->insert( "//depot/main/...", "//client/..." );

// called with one argument containing both halves of the mapping:
$map->insert( "//depot/live/... //client/live/..." );

// called with one argument containing a half-map:
// This call produces the mapping "depot/... depot/..."
$map->insert( "depot/..." );
```

$map->translate ( string, [ bool ]) -> string

Translate a string through a map, and return the result. If the optional second argument is 1, translate forward, and if it is 0, translate in the reverse direction. By default, translation is in the forward direction.

$map->includes( string ) -> bool

Tests whether a path is mapped or not.

```php
if $map->includes( "//depot/main/..." ) {
    ...
}
```

$map->reverse() -> P4_Map

Return a new P4_Map object with the left and right sides of the mapping swapped. The original object is unchanged.

$map->lhs() -> array

Returns the left side of a mapping as an array.

$map->rhs() -> array

Returns the right side of a mapping as an array.
$map->as_array() -> array

Returns the map as an array.
Class P4_MergeData

Description

Class containing the context for an individual merge during execution of a `p4 resolve`.

Properties

$md->your_name -> string

Returns the name of "your" file in the merge. This is typically a path to a file in the workspace.

$md->their_name -> string

Returns the name of "their" file in the merge. This is typically a path to a file in the depot.

$md->base_name -> string

Returns the name of the "base" file in the merge. This is typically a path to a file in the depot.

$md->your_path -> string

Returns the path of "your" file in the merge. This is typically a path to a file in the workspace.

$md->their_path -> string

Returns the path of "their" file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `their_name` have been loaded.

$md->base_path -> string

Returns the path of the base file in the merge. This is typically a path to a temporary file on your local machine in which the contents of `base_name` have been loaded.

$md->result_path -> string

Returns the path to the merge result. This is typically a path to a temporary file on your local machine in which the contents of the automatic merge performed by the server have been loaded.

$md->merge_hint -> string

Returns the hint from the server as to how it thinks you might best resolve this merge.
Class P4_OutputHandlerAbstract

Description

The P4_OutputHandlerAbstract class is a handler class that provides access to streaming output from the server. After defining the output handler, set $p4->handler to an instance of a subclass of P4_OutputHandlerAbstract.

By default, P4_OutputHandlerAbstract returns HANDLER_REPORT for all output methods. The different return options are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANDLER_REPORT</td>
<td>Messages added to output (don’t handle, don’t cancel).</td>
</tr>
<tr>
<td>HANDLER.Handled</td>
<td>Output is handled by class (don’t add message to output).</td>
</tr>
<tr>
<td>HANDLER_Cancel</td>
<td>Operation is marked for cancel, message is added to output.</td>
</tr>
</tbody>
</table>

Class Methods

class MyHandler extends P4_OutputHandlerAbstract

Constructs a new subclass of P4_OutputHandlerAbstract.

Instance Methods

$handler->outputBinary -> int

Process binary data.

$handler->outputInfo -> int

Process tabular data.

$handler->outputMessage -> int

Process informational or error messages.

$handler->outputStat -> int

Process tagged data.

$handler->outputText -> int

Process text data.
Class P4_Resolver

Description

P4_Resolver is a class for handling resolves in Perforce. It must be subclassed, to be used; subclasses
 can override the P4::resolve() method. When P4::run_resolve() is called with a P4_Resolver object,
 it calls the P4_Resolver::resolve() method of the object once for each scheduled resolve.

Properties

None.

Static Methods

None.

Instance Methods

$resolver->resolve( self, mergeData ) => string

Returns the resolve decision as a string. The standard Perforce resolve strings apply:

<table>
<thead>
<tr>
<th>String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ay</td>
<td>Accept Yours.</td>
</tr>
<tr>
<td>at</td>
<td>Accept Theirs.</td>
</tr>
<tr>
<td>am</td>
<td>Accept Merge result.</td>
</tr>
<tr>
<td>ae</td>
<td>Accept Edited result.</td>
</tr>
<tr>
<td>s</td>
<td>Skip this merge.</td>
</tr>
<tr>
<td>q</td>
<td>Abort the merge.</td>
</tr>
</tbody>
</table>

By default, all automatic merges are accepted, and all merges with conflicts are skipped. The
P4_Resolver::resolve() method is called with a single parameter, which is a reference to a
P4_MergeData object.
Perforce software includes software developed by the University of California, Berkeley and its contributors. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).