ORACLE

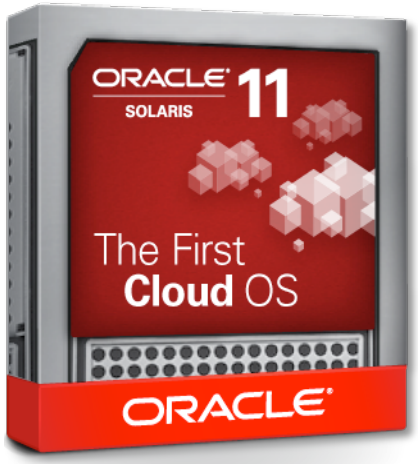
SOLARIS

Solaris 11 Developer Webinar Series
Webinar #7

Publishing IPS Packages – Guide for Developers

Eric Reid
Oracle Systems ISV Engineering

Brock Pytlik
Oracle Solaris Engineering



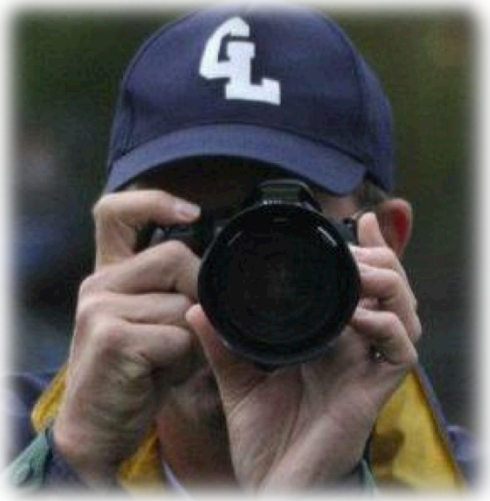
Solaris 11 for Developers Webinar Series

ORACLE SOLARIS

Webinar Series Topic	Date	Speaker
Modern Software Packaging for Enterprise Developers	03-27-12 @ 9am PT	Eric Reid
Simplify Your Development Environment with Zones, ZFS & More	04-10-12 @ 9am PT	Eric Reid & Stefan Schneider
Managing Application Services – Using SMF Manifests in Solaris 11	04-24-12 @ 9am PT	Matthew Hosanee
Optimize Your Applications on Solaris 11: The DTrace Advantage	05-08-12 @ 9am PT	Angelo Rajadurai
Maximize Application Performance and Reliability on Solaris 11	05-22-12 @ 9am PT	Vijay Taktar
Writing Solaris 11 Device Drivers	06-05-12 @ 9am PT	Bill Knoche
Publishing IPS Packages	06-19-12 @ 9am PT	Eric Reid & Brock Pytlik
The World of IPS, Part 2: Tools and Commands	07-17-12 @ 9am PT	Eric Reid & Brock Pytlik
Oracle Solaris Remote Lab	07-31-12 @ 9am PT	Ron Larson & Angelo Rajadurai

Eric Reid
Oracle Systems ISV Engineering

- 24-year Sun/Oracle employee
- Has worked with SunOS/ Solaris/OpenSolaris since 1985
- Home-based employee for Sun/ Oracle since 1996
- Oracle Systems ISV Engineering works closely with our most important Systems and Solaris partners



Yes, I enjoy digital photography…

Brock Pytlik
Oracle Solaris Engineering

- Started at Oracle in 2008
- Has worked on IPS for the last 4 years.

What You'll Learn Today

- IPS Concepts and Terms
- IPS Developer Tools and Commands
- Architectural Considerations when developing IPS Packages
- Creating a Simple IPS Package
- Publishing an IPS Package

Assumption for this Webinar: some knowledge of IPS concepts and consumer-level commands (`pkg(1)`, etc.)

Next webinar (17 July): Advanced IPS topics, including how to implement package 'scripting'

Best Resource for IPS Package Developers

- IPS technology page

http://www.oracle.com/technetwork/server-storage/solaris11/technologies/ips-323421.html

- Whitepaper: *Packaging and Delivering Software with the Image Packaging System: A developer's guide*

http://hub.opensolaris.org/bin/download/Project+pkg/files/ipsdevguide.pdf

- Oracle University Course (Instructor-led or Virtual): *Developing and Deploying Applications on Oracle Solaris 11*

http://education.oracle.com/pls/web_prod-plq-dad/db_pages.getCourseDesc?dc=D73486GC10

Why Should Developers Care About IPS?

- Supported, consistent installation mechanism
  - ...with tools!
- Dependency checking
  - Automatically integrated with OS and other 3rd party packages
- Transparent multi-architecture support
- Fast
- Secure
- Zone support
- Full Solaris Boot Environment support
- ~~Patch~~ Update support
- Standalone or repository (network) install
- One-click install from the web
- Integrated with Solaris Automated Installer
- Integrated with Oracle Enterprise Manager Ops Center

IPS Concepts and Terms - 1

- Action
  - The atomic units of software delivery into IPS
  - Types: `file, directory, link, set, driver, depend, license, user, legacy, signature`
- Manifest
  - A file which describes a collection of Actions
- Package
  - A set of Actions to be taken upon installation, upgrade or removal; described by a Manifest
- Repository
  - A collection of Packages, accessible via HTTP or local filesystem
- Publisher
  - An entity that develops and constructs Packages

IPS Concepts and Terms - 2

- Variant
  - Mechanism to allow a package to be used in multiple situations (multiple CPU architecture; debug/non-debug bits; Global Zone bits vs non-Global Zone bits)
- Facet
  - Mechanism to create optional portions of packages (for locales, man pages, etc.)
- Actuator
  - Tag applied to any action that causes a system change when that action is installed, removed or updated
- Package FMRI
  - Unique Fault Management Resource Identifer
  - Example:

```
pkg://solaris/system/library@0.5.11,5.11-0.175.0.0.0.2.1:20111019T082311Z
```

Publisher Category/Name Version Creation Timestamp

IPS Concepts and Terms - 3

- For Developers:
  - Actions + Manifest ➔ Package
  - Packages ➔ Repository
    Package ➔ Package Archive file (.p5p)
  - Multiple versions of Packages can coexist in Repositories

- For Consumers (target systems):
  - Only one version of a package may be installed at a time
  - Accessing Packages via Repositories:
    - Consumer System subscribes to one or more Publisher's Repositories
    - Packages are identified by use of pkg:// FMRIs
  - Installing Package Archive files:
    - Outside the bounds of Repositories

IPS Developer Tools and Commands - 1 (pkg:/package/pkg)

- `pkgsend(1)`: Publish/update packages
  - `generate` subcommand
    - Create manifest from files/paths/SVR4 package
  - `publish` subcommand
    - Publish package
- `pkgmogrify(1)`: Programmatically edit manifests
- `pkgdepend(1)`: Manage package dependencies
  - `generate` subcommand
    - Generate dependencies for a package by examining package content
  - `resolve` subcommand
    - Resolves package dependencies by examining other packages

IPS Developer Tools and Commands - 2 (pkg:/package/pkg)

- `pkgmerge(1)`: Create multi-variant packages
- `pkglint(1)`: 'Sanity check' one or more package manifests
- `pkgsign(1)`: Cryptographically sign a package
- `pkgrepo(1)`: Create and manage packages repositories
- `pkgfmt(1)`: Make more human-readable manifests

Package Dependencies

- Dependencies are not optional!
- Require: Causes another package to be installed
- Require-any: Installs one of a set of packages
- Conditional: If one package is installed, install a second
- Group: A require that a user can override
- Optional: Establishes a minimum version
- Origin: Establishes a minimum version in the current image
- Incorporate: Constrain the version of another package
- Parent: Incorporate enforced on non-global zones
- Exclude: Prevent another package from being installed

General guidelines

- Fixes or features (any change) means a new version of a package
- Package obsoletion – rarely a good idea
  - Upgrade is prevented if any packages depend on the package
  - Usually a better idea to let the package continue to exist
- Moving files between packages
  - Use an optional dependency to ensure that two packages don't think they both own a path at the same time.
  - If it's an editable file, set the original_name attribute
- Renaming packages – sometimes necessary
  - Old names persist and are mapped accordingly

How to distribute packages

- Network-based package repositories
  - Simplest for customers
  - Useful for diverse customer base
  - Put apache in front of pkg.depotd
- File-based package repositories
  - Typically faster than network-based repositories
  - Useful for internal deployment
- Package archives
  - Useful for disconnected customers
  - Require customers to retrieve archives manually

Creating a Simple IPS Package Planning

- The package creation and publication ‘assembly line’
  1. Lay out your package as you want it – the *proto area*
  2. Create initial manifest from *proto area* with `pkgsend generate`
  3. Modify generated manifest: `pkgmogrify` Step 2 manifest.
     - Add package name, facets, actuators, description, summary, …
     - Modify file layout, permissions, owners …
  4. Evaluate packages dependencies: `pkgdepend generate` on Step 3 manifest
  5. Resolve packages dependencies `pkgdepend resolve` on all Step 4 manifests
  6. Verify: `pkglint` Step 5 manifests to catch any errors
  7. (Optional) Merge: `pkgmerge` the package from different repositories to create multi-variant packages

Creating a Simple IPS Package Example

- We wish to create a package called *mysoftware,* which delivers files under `/opt`
- This package consists of a library, binary, and optional man page
- The library and the binary both depend on the system libc.so.1 library
- The *mysoftware* package will deliver the following files to the system:

```
/opt/mysoftware/lib/mylib.so.1
/opt/mysoftware/bin/mycmd
/opt/mysoftware/man/man1/mycmd.1
```

Creating a Simple IPS Package 'Assembly Line'

1. Lay out your package as you want it – the *proto area*

We'll be working in /proto as a non-privileged user

```
/proto/opt/mysoftware/lib/mylib.so.1
/proto/opt/mysoftware/bin/mycmd
/proto/opt/mysoftware/man/man1/mycmd.1
```

Creating a Simple IPS Package 'Assembly Line'

2. Create initial manifest from *proto area* with `pkgsend generate`

We use `pkgfmt` to make things more readable.

```
$ pkgsend generate proto | pkgfmt > mypkg.p5m.gen
```

This first intermediate file looks like this:

```
dir path=opt group=bin mode=0755 owner=root
dir path=opt/mysoftware group=bin mode=0755 owner=root
dir path=opt/mysoftware/bin group=bin mode=0755 owner=root
dir path=opt/mysoftware/lib group=bin mode=0755 owner=root
dir path=opt/mysoftware/man group=bin mode=0755 owner=root
dir path=opt/mysoftware/man/man1 group=bin mode=0755 owner=root
file opt/mysoftware/bin/mycmd path=opt/mysoftware/bin/mycmd group=bin \
    mode=0755 owner=root
file opt/mysoftware/lib/mylib.so.1 path=opt/mysoftware/lib/mylib.so.1 \
    group=bin mode=0644 owner=root
file opt/mysoftware/man/man1/mycmd.1 path=opt/mysoftware/man/man1/mycmd.1 \
    group=bin mode=0644 owner=root
```

Creating a Simple IPS Package 'Assembly Line'

3. Generate required metadata: `pkgmogrify` Step 2 manifest

We create `mypkg.mog` with metadata to be integrated into the manifest:

```
set name=pkg.fmri value=mypkg@1.0,5.11-0
set name=pkg.summary value="This is our example package"
set name=pkg.description value="This is a full description of \
all the interesting attributes of this example package."
set name=variant.arch value=$(ARCH)
set name=info.classification \
    value=org.opensolaris.category.2008:Applications/Accessories
link path=usr/share/man/index.d/mysoftware target=opt/mysoftware/man
<transform dir path=opt$->drop>
<transform dir file link hardlink path=opt/.+/man(/.+)? -> \
    default facet.doc.man true>
```

We then use pkgmogrify to pull it all together:

```
$ pkgmogrify -DARCH=`uname -p` mypkg.p5m.gen mypkg.mog  | pkgfmt >
mypkg.p5m.mog
```

Creating a Simple IPS Package 'Assembly Line'

4. Evaluate packages dependencies: `pkgdepend generate` on Step 3 manifest

5. Resolve packages dependencies `pkgdepend resolve` on all Step 4 manifests

```
$ pkgdepend generate -md proto mypkg.p5m.mog > mypkg.p5m.dep
$ pkgdepend resolve -m mypkg.p5m.dep
```

Creating a Simple IPS Package 'Assembly Line'

6. Verify: `pkglint` manifests to catch any errors

```
$ pkglint mypkg.p5m.4.res
Lint engine setup…
Starting lint run…
```

7. (Optional): Merge: `pkgmerge` the package from different repositories to create multi-variant packages.

Create a package that:

- Installs on both Sparc and x86
- Includes debug and non-debug bits
- Works in both the global zone and non-global zone

Publishing a Simple IPS Package Deployment

- For development, publishing to a private file repository is best
- For deployment, the choice of the type of repository depends on business and customer needs

Publishing a Simple IPS Package
To local file-based repository

Create the repo under `/scratch`

```
$ pkgrepo create /scratch/my-repository
$ pkgrepo -s /scratch/my-repository set publisher/prefix=mypublisher
$ find /scratch/my-repository/
/scratch/my-repository/
/scratch/my-repository/pkg5.repository
```

Publish the package to the repo, then examine the repo

```
$ pkgsend -s /scratch/my-repository/ publish -d proto mypkg.p5m.4.res
pkg://mypublisher/mypkg@1.0,5.11-0:20120619T034303Z
PUBLISHED
$ pkgrepo -s /scratch/my-repository info
PUBLISHER   PACKAGES STATUS           UPDATED
mypublisher 1        online           2012-06-13T03:43:04.117536Z
```

If desired, this repo can also be served over the network using `pkg.depotd(1M)`. Use an apache proxy if this repository will have much load.

Publishing a Simple IPS Package
Testing the package - 1

As a user with Software Installation privileges (or root) configure the publisher

```
$ sudo pkg set-publisher -p /scratch/my-repository
pkg set-publisher:
  Added publisher(s): mypublisher
```

Install the package:

```
$ sudo pkg install mypkg
           Packages to install:  1
       Create boot environment: No
Create backup boot environment: No
DOWNLOAD                                  PKGS       FILES    XFER (MB)
Completed                                  1/1         3/3      0.7/0.7
PHASE                                        ACTIONS
Install Phase                                  15/15
PHASE                                          ITEMS
Package State Update Phase                       1/1
Image State Update Phase                         2/2
PHASE                                          ITEMS
Reading Existing Index                           8/8
Indexing Packages                                1/1
```

Publishing a Simple IPS Package
Testing the package - 2

Check that the files expected are present and that the metadata is correct.

```
$ find /opt/mysoftware/
/opt/mysoftware/
/opt/mysoftware/bin
/opt/mysoftware/bin/mycmd
/opt/mysoftware/lib
/opt/mysoftware/lib/mylib.so.1
/opt/mysoftware/man
       :

$ pkg info mypkg
          Name: mypkg
       Summary: This is our example package
   Description: This is a full description of all the interesting attributes of
                this example package.
      Category: Applications/Accessories
         State: Installed
     Publisher: mypublisher
       Version: 1.0
       :
```

Resources for Solaris 11 Developers

Recorded Sessions

Webinar Series Topic	Date	Speaker
Writing Oracle Solaris 11 Device Drivers Replay - Slides ✔	06-05-12 @ 9am PT	Bill Knoche (Principal Software Engineer)
Maximize Application Performance and Reliability with Oracle Solaris Studio Replay - Slides ✔	05-22-12 @ 9am PT	Vijay Tatkar (Senior Manager Software Development)
Optimize Your Applications on Oracle Solaris 11: The DTrace Advantage Replay - Slides ✔	05-08-12 @ 9am PT	Angelo Rajadurai (Principal Software Engineer)
Managing Application Services - Using SMF Manifests in Oracle Solaris 11 Replay - Slides ✔	04-24-12 @ 9am PT	Matthew Hosanee (Principal Software Engineer)
Simplify Your Development Environement with Zones, ZFS & More Replay - Slides ✔	04-10-12 @ 9am PT	Eric Reid (Principal Software Engineer) and Stefan Schneid[illegible] (Chief Technolog[illegible] ISV-Engineerin[illegible])
Modern Software Packaging for Enterprise Developers Replay - Slides ✔	03-27-12 @ 9am PT	Eric Re[illegible] (Principal S[illegible]ftware Engi[illegible]er)

Agenda - Next Sessions

- Click on event to register
- All webinars on Tuesday's 9-10am PT (Event will support VOIP)

Webinar Series Topic	Date	Speaker
Publishing IPS Packages	06-19-12 @ 9am PT	Eric Reid (Principal Software Engineer) and Brock Pytlik (Senior Software Engineer
The World of IPS 2: Tools and Commands	07-17-12 @ 9am PT	Eric Reid (Principal Software Engineer) and Brock Pytlik (Senior Software Engineer
Oracle Solaris Remote Lab	07-31-12 @ 9am PT	Ron Larson (Project Manager, ISV Engineering) and Angelo Rajadurai (Principal Software Engineer)

Today's Webinar

Registration:
http://www.oracle.com/technetwork/server-storage/solaris11/overview/webinar-series-1563626.html

Questions

Hardware and Software
Engineered to Work Together

Publishing a Simple IPS Package Alternative: To Package Archive (.p5p)

```
$ pkgrecv -s /scratch/my-repository -a -d myarchive.p5p mypkg
Retrieving packages for publisher mypublisher …
Retrieving and evaluating 1 package(s)…
DOWNLOAD                                  PKGS       FILES    XFER (MB)
Completed                                  1/1         3/3      0.7/0.7
ARCHIVE                                                FILES   STORE (MB)
myarchive.p5p                                          14/14      0.7/0.7
```

Tips and tricks: pkgsend

- Make your build produce a proto area which mirrors how the software should be installed.
- Use -T with files where timestamps matter
    - .py files generated .pyc files if the .py file is older than .pyc files
- Use --target if your proto area has hardlinks for reproducible packaging operations.

Tips and tricks: pkgdepend resolve

- Use -R to decouple build machine from packaging results
  - Resolve usually uses resolves against installed packages
  - -S is an option, but usually leads to unresolved dependencies
  - Create standard reference images and resolve against them
    - `pkgdepend -R <path to image> resolve ...`