AppStream & Listaller

Matthias Klumpp
mak@debian.org
matthias@tenstral.net

February 2014

Who am I?

- PackageKit Developer
- Debian Developer
- Contributor to KDE, GNOME
- AppStream and Listaller maintainer

Outline of the Talk

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
    - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
    - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
    - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution’s software repositories is displayed in form of packages
  - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
  - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
  - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
  - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
  - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
  - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
  - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
  - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
  - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
    - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
    - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
    - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
    - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
    - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
    - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application management?

- Content of distribution's software repositories is displayed in form of packages
  - Unclear for non-technical users: What is a package? Why are there so many of them?
- Existing software-centers are distribution specific and not well integrated with their desktop environments
- Applications are not presented well to the user
  - localization missing, bad or no screenshots, missing urls, inconsistencies between distributions, ...
- No interaction possible
  - We want user ratings, reviews, maybe an easy way to report bugs in the software-center, etc.

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

What is wrong with application distribution?

- PPAs/Repositories are insecure: 3rd-party applications are installed with root permission, may override system components, break distribution upgrades, ...
- Handling of PPAs is not very simple for users
- PPAs are distribution-specific: Many 3rd-party apps are available for e.g. Ubuntu, while other distributions need their own repos
- PPAs have a complex structure and are an overkill if people just want to distribute an app on Linux
- Binary installers don't integrate well and are difficult to handle. If executed as root, they are a potential security risk

We need a simple and secure cross-desktop solution to install 3rd-party applications, which integrates well with the rest of the system

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance "component metadata" in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance “component metadata” in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance “component metadata” in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance “component metadata” in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance "component metadata" in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance "component metadata" in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance “component metadata” in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

Two independent projects

AppStream

- Cross-distro specifications for building software-center applications
- Metadata / database specs for distributors
- Some (optional) new metadata for upstream projects
- Interactive features (Ratings & Reviews, ...)

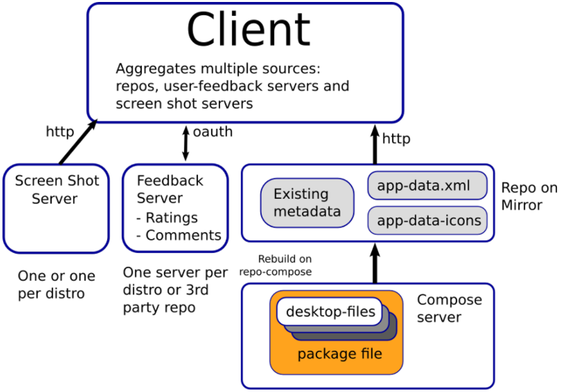
Listaller

- Generates cross-distro application packages
- Tools for app developers to make their app work on many distros
- Additional specs to enhance "component metadata" in distributions
- Optional component, requires PackageKit (and ideally AppStream) to work

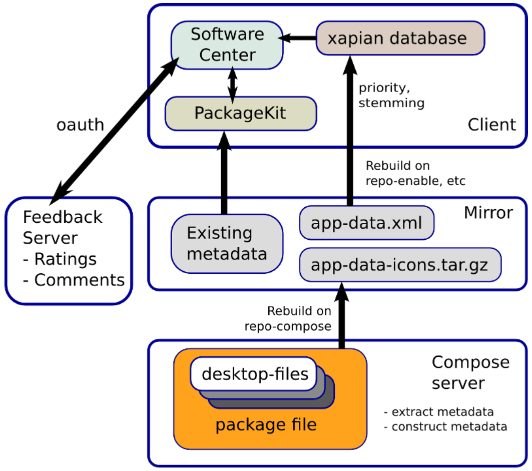
AppInstaller meeting 2011

Fedora, Debian, OpenSUSE, Mageia, Ubuntu, KDE, Freedesktop

AppStream Concept



AppStream Concept



AppStream XML

AppStream XML contains meta-data for each application, such as name,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file),

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords, screenshot references and descriptions,

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords, screenshot references and descriptions, etc.

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords, screenshot references and descriptions, etc.
It also allows localization of some data

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords, screenshot references and descriptions, etc.
It also allows localization of some data

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppStream XML

AppStream XML contains meta-data for each application, such as name, unique app-id (desktop file), summary, description, icon, author data, project group, categories, mimetypes, keywords, screenshot references and descriptions, etc.
It also allows localization of some data

XML!?

Debian FTP-Masters prefer YAML, adding XML to the metadata set is discouraged. A more general solution is wanted too.

Problem

Where do we get the data from if the distribution does not have it already in it's package metadata or it is missing from desktop-files? Where do we get good screenshots from?

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

AppData

- Small XML file shipped with upstream project
- Subset of the AppStream XML specification
- Contains screenshot URLs to 3rd-party servers
- Is localized upstream
- **No requirement** to implement AppStream! It is just additional metadata to extend or improve existing data
- We would like to have upstreams ship the file to improve the quality of data which is used to present apps in a software center
- External content (like screenshots) is cached and verified on the distribution's server

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

Some metadata statistics

Fedora ships AppStream since v20

Applications in Fedora with:

- Long descriptions: 170 (9%)
- Keywords: 95 (5%)
- Categories: 1744 (98%)
- Screenshots: 143 (8%)

- GNOME applications with AppData: 60 (50%)
- KDE applications with AppData: 1 (1%)
- XFCE applications with AppData: 0 (0%)

http://alt.fedoraproject.org/pub/alt/screenshots/f20/status.html

One database to rule them all

- In order to make use of AppStream, you have to combine many data sources (and consider fallback solutions)
- Debian will provide AppStream data in YAML (describing not only applications, but providing extra archive metadata), Ubuntu uses an own desktop-file extension, others use XML
- The AppStream software and libappstream were created to avoid to require every software center to provide parsers to all formats
- A Xapian database is generated, containing all data for software centers to access
- libappstream allows acessing the database and various other data sources using a GObject-based API (you don't have to work with Xapian's C++ interface)

One database to rule them all

- In order to make use of AppStream, you have to combine many data sources (and consider fallback solutions)
- Debian will provide AppStream data in YAML (describing not only applications, but providing extra archive metadata), Ubuntu uses an own desktop-file extension, others use XML

- The AppStream software and libappstream were created to avoid to require every software center to provide parsers to all formats
- A Xapian database is generated, containing all data for software centers to access
- libappstream allows acessing the database and various other data sources using a GObject-based API (you don't have to work with Xapian's C++ interface)

One database to rule them all

- In order to make use of AppStream, you have to combine many data sources (and consider fallback solutions)
- Debian will provide AppStream data in YAML (describing not only applications, but providing extra archive metadata), Ubuntu uses an own desktop-file extension, others use XML

- The AppStream software and libappstream were created to avoid to require every software center to provide parsers to all formats
- A Xapian database is generated, containing all data for software centers to access
- libappstream allows acessing the database and various other data sources using a GObject-based API (you don't have to work with Xapian's C++ interface)

One database to rule them all

- In order to make use of AppStream, you have to combine many data sources (and consider fallback solutions)
- Debian will provide AppStream data in YAML (describing not only applications, but providing extra archive metadata), Ubuntu uses an own desktop-file extension, others use XML

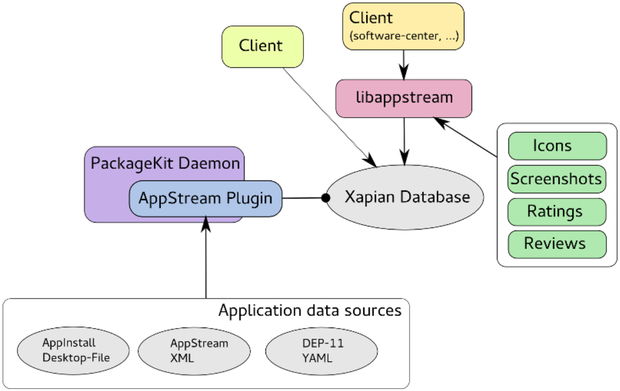
- The AppStream software and libappstream were created to avoid to require every software center to provide parsers to all formats
- A Xapian database is generated, containing all data for software centers to access
- libappstream allows acessing the database and various other data sources using a GObject-based API (you don't have to work with Xapian's C++ interface)

One database to rule them all

- In order to make use of AppStream, you have to combine many data sources (and consider fallback solutions)
- Debian will provide AppStream data in YAML (describing not only applications, but providing extra archive metadata), Ubuntu uses an own desktop-file extension, others use XML

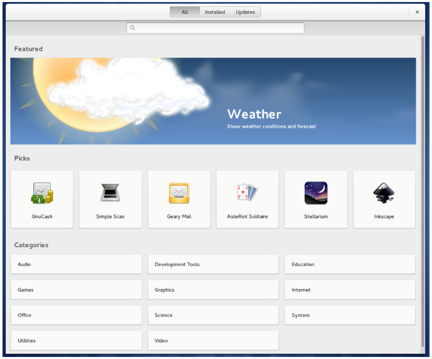
- The AppStream software and libappstream were created to avoid to require every software center to provide parsers to all formats
- A Xapian database is generated, containing all data for software centers to access
- libappstream allows acessing the database and various other data sources using a GObject-based API (you don't have to work with Xapian's C++ interface)

One database to rule them all



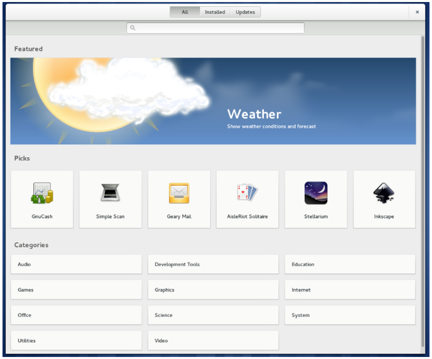
GNOME? - AppStream status in desktops and distros

- GNOME-Software: Implementation of an AppStream-compatible software-center by the GNOME project
- Fully supported on Fedora, pending on other distributions, blocked on Debian by DEP-11 implementation



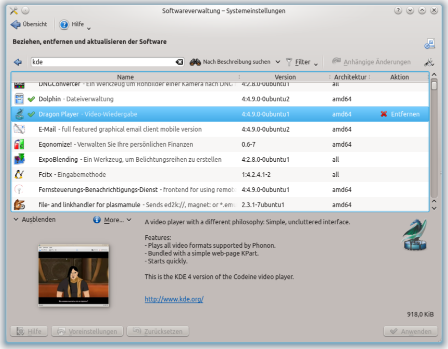
GNOME? - AppStream status in desktops and distros

- GNOME-Software: Implementation of an AppStream-compatible software-center by the GNOME project
- Fully supported on Fedora, pending on other distributions, blocked on Debian by DEP-11 implementation



KDE? - AppStream status in desktops and distros

- Apper: Initial support for AppStream via libappstream, needs work
- AppData inclusion into KDE projects is currently discussed
- Possible port of Muon Discover to PackageKit and AppStream



Name	Version	Architektur	Aktion
DNGConverter - Ein Werkzeug um Rohbilder einer Kamera nach DNG	4:2.8.0-0ubuntu1	all	
Dolphin - Dateiverwaltung	4:4.9.0-0ubuntu2	amd64	
Dragon Player - Video-Wiedergabe	4:4.9.0-0ubuntu1	amd64	Entfernen
E-Mail - full featured graphical email client mobile version	4:4.9.0-0ubuntu2	amd64	
Eqonomize! - Verwalten Sie Ihre persönlichen Finanzen	0.6-7	amd64	
ExpoBlending - Ein Werkzeug, um Belichtungsreihen zu erstellen	4:2.8.0-0ubuntu1	all	
Fcitx - Eingabemethode	1:4.2.4.1-2	all	
Fernsteuerungs-Benachrichtigungs-Dienst - frontend for using remot	4:4.9.0-0ubuntu1	amd64	
file- and linkhandler for plasmamule - Sends ed2k://, magnet: or *.emu	2.3.1-7ubuntu1	amd64	

A video player with a different philosophy: Simple, uncluttered interface.

Features:
- Plays all video formats supported by Phonon.
- Bundled with a simple web-page KPart.
- Starts quickly.

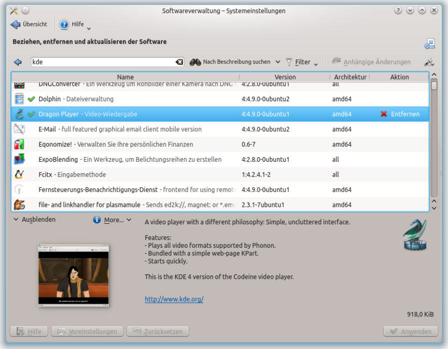
This is the KDE 4 version of the Codeine video player.

http://www.kde.org/

918,0 KiB

KDE? - AppStream status in desktops and distros

- Apper: Initial support for AppStream via libappstream, needs work
- AppData inclusion into KDE projects is currently discussed
- Possible port of Muon Discover to PackageKit and AppStream



Beziehen, entfernen und aktualisieren der Software

kde

Name	Version	Architektur	Aktion
DNGConverter - Ein Werkzeug um Rohbilder einer Kamera nach DNG	4:2.8.0-0ubuntu1	all	
✓ Dolphin - Dateiverwaltung	4:4.9.0-0ubuntu2	amd64	
✓ Dragon Player - Video-Wiedergabe	4:4.9.0-0ubuntu1	amd64	✗ Entfernen
E-Mail - full featured graphical email client mobile version	4:4.9.0-0ubuntu2	amd64	
Eqonomize! - Verwalten Sie Ihre persönlichen Finanzen	0.6-7	amd64	
ExpoBlending - Ein Werkzeug, um Belichtungsreihen zu erstellen	4:2.8.0-0ubuntu1	all	
Fcitx - Eingabemethode	1:4.2.4.1-2	all	
Fernsteuerungs-Benachrichtigungs-Dienst - frontend for using remote	4:4.9.0-0ubuntu1	amd64	
file- and linkhandler for plasmamule - Sends ed2k://, magnet: or *.emu	2.3.1-7ubuntu1	amd64	

A video player with a different philosophy: Simple, uncluttered interface.

Features:
- Plays all video formats supported by Phonon.
- Bundled with a simple web-page KPart.
- Starts quickly.

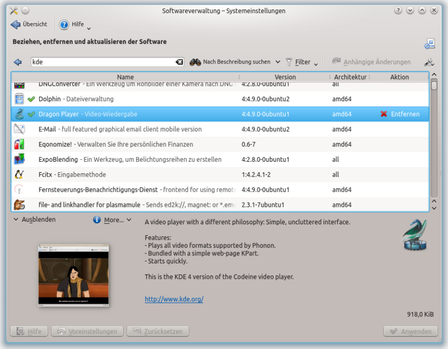
This is the KDE 4 version of the Codeine video player.

http://www.kde.org/

918,0 KiB

KDE? - AppStream status in desktops and distros

- Apper: Initial support for AppStream via libappstream, needs work
- AppData inclusion into KDE projects is currently discussed
- Possible port of Muon Discover to PackageKit and AppStream



Name	Version	Architektur	Aktion
DNGConverter - Ein Werkzeug um Rohbilder einer Kamera nach DNG	4:2.8.0-0ubuntu1	all	
✓ Dolphin - Dateiverwaltung	4:4.9.0-0ubuntu2	amd64	
✓ Dragon Player - Video-Wiedergabe	4:4.9.0-0ubuntu1	amd64	Entfernen
E-Mail - full featured graphical email client mobile version	4:4.9.0-0ubuntu2	amd64	
Eqonomize! - Verwalten Sie Ihre persönlichen Finanzen	0.6-7	amd64	
ExpoBlending - Ein Werkzeug, um Belichtungsreihen zu erstellen	4:2.8.0-0ubuntu1	all	
Fcitx - Eingabemethode	1:4.2.4.1-2	all	
Fernsteuerungs-Benachrichtigungs-Dienst - frontend for using remote	4:4.9.0-0ubuntu1	amd64	
file- and linkhandler for plasmamule - Sends ed2k://, magnet: or *.emu	2.3.1-7ubuntu1	amd64	

A video player with a different philosophy: Simple, uncluttered interface.

Features:
- Plays all video formats supported by Phonon.
- Bundled with a simple web-page KPart.
- Starts quickly.

This is the KDE 4 version of the Codeine video player.

http://www.kde.org/

918,0 KiB

AppStream status in desktops and distros

- Debian: Work on the universal component-metadata file (defined as DEP-11) has started
- Ubuntu: Ships AppInstall, which can be consumed by libappstream, but will likely migrate to Debian's solution
- Fedora: Ships AppStream XML and GNOME-Software, already supports screenshots
- OpenSUSE: Works on AppStream XML, work finished?
- So far, only GNOME has a software-center which fully supports almost everything from the AppStream specification

AppStream status in desktops and distros

- Debian: Work on the universal component-metadata file (defined as DEP-11) has started
- Ubuntu: Ships AppInstall, which can be consumed by libappstream, but will likely migrate to Debian's solution
- Fedora: Ships AppStream XML and GNOME-Software, already supports screenshots
- OpenSUSE: Works on AppStream XML, work finished?
- So far, only GNOME has a software-center which fully supports almost everything from the AppStream specification

AppStream status in desktops and distros

- Debian: Work on the universal component-metadata file (defined as DEP-11) has started
- Ubuntu: Ships AppInstall, which can be consumed by libappstream, but will likely migrate to Debian's solution
- Fedora: Ships AppStream XML and GNOME-Software, already supports screenshots
- OpenSUSE: Works on AppStream XML, work finished?
- So far, only GNOME has a software-center which fully supports almost everything from the AppStream specification

AppStream status in desktops and distros

- Debian: Work on the universal component-metadata file (defined as DEP-11) has started
- Ubuntu: Ships AppInstall, which can be consumed by libappstream, but will likely migrate to Debian's solution
- Fedora: Ships AppStream XML and GNOME-Software, already supports screenshots
- OpenSUSE: Works on AppStream XML, work finished?
- So far, only GNOME has a software-center which fully supports almost everything from the AppStream specification

AppStream status in desktops and distros

- Debian: Work on the universal component-metadata file (defined as DEP-11) has started
- Ubuntu: Ships AppInstall, which can be consumed by libappstream, but will likely migrate to Debian's solution
- Fedora: Ships AppStream XML and GNOME-Software, already supports screenshots
- OpenSUSE: Works on AppStream XML, work finished?
- So far, only GNOME has a software-center which fully supports almost everything from the AppStream specification

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Listaller?

Listaller is a complete solution for packaging and distributing 3rd-party applications. It provides tools for building cross-distro applications, creating and signing packages, update management and related features. It is completely invisible to the user on any system using PackageKit.

- Started in 2008 as an experiment
  - Covered features of PackageKit and AppStream
  - Switched to PackageKit in 2009
- Merged with Autopackage and some other projects in 2010
- AppStream was started in 2011, in turn Listaller was rewritten from scratch, dropping duplicate functionality
- Rewrite finished in 2012, many new concepts were implemented in 2013

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

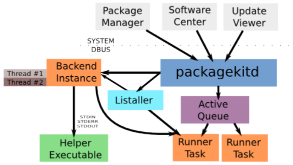
- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

Goals

- System integration
  - Users should not notice that Listaller is used when installing apps
  - Software updates should be retrieved using the same UI as the system itself
  - Listaller apps should integrate seamlessly with the system
- Cross-distro and -desktop compatibility
- Simplification
  - No catch-all solution, Listaller should cover the most common use-cases. Native distribution packages should cover the remaining cases.
- Security
  - Signatures, security hints database, sandboxing, ...
- Developer tools
  - Provide helpers for developers to make their apps run on multiple distributions
  - Make packaging as simple as possible, do some QA on the packaged app

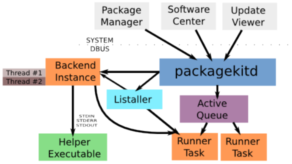
Basic concept

- Listaller contains a PackageKit plugin, mediating between Listaller and PackageKit
- The plugin acts as »meta-backend«, sending information about Listaller packages via PackageKit's DBus interface, and making queries to the native backend
    - Every PackageKit client can install, remove and update Listaller packages
- Listaller installs some XML for AppStream-compatible software centers, so they can display details about a 3rd-party application
- Uses a superset of the AppData specification as source for application metadata



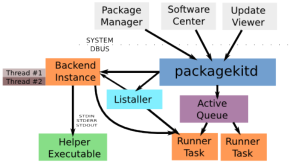
Basic concept

- Listaller contains a PackageKit plugin, mediating between Listaller and PackageKit
- The plugin acts as »meta-backend«, sending information about Listaller packages via PackageKit's DBus interface, and making queries to the native backend
    - Every PackageKit client can install, remove and update Listaller packages
- Listaller installs some XML for AppStream-compatible software centers, so they can display details about a 3rd-party application
- Uses a superset of the AppData specification as source for application metadata



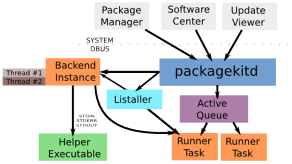
Basic concept

- Listaller contains a PackageKit plugin, mediating between Listaller and PackageKit
- The plugin acts as »meta-backend«, sending information about Listaller packages via PackageKit's DBus interface, and making queries to the native backend
  - Every PackageKit client can install, remove and update Listaller packages
- Listaller installs some XML for AppStream-compatible software centers, so they can display details about a 3rd-party application
- Uses a superset of the AppData specification as source for application metadata



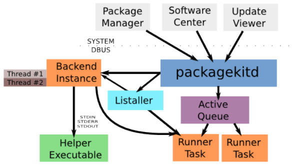
Basic concept

- Listaller contains a PackageKit plugin, mediating between Listaller and PackageKit
- The plugin acts as »meta-backend«, sending information about Listaller packages via PackageKit's DBus interface, and making queries to the native backend
  - Every PackageKit client can install, remove and update Listaller packages
- Listaller installs some XML for AppStream-compatible software centers, so they can display details about a 3rd-party application
- Uses a superset of the AppData specification as source for application metadata

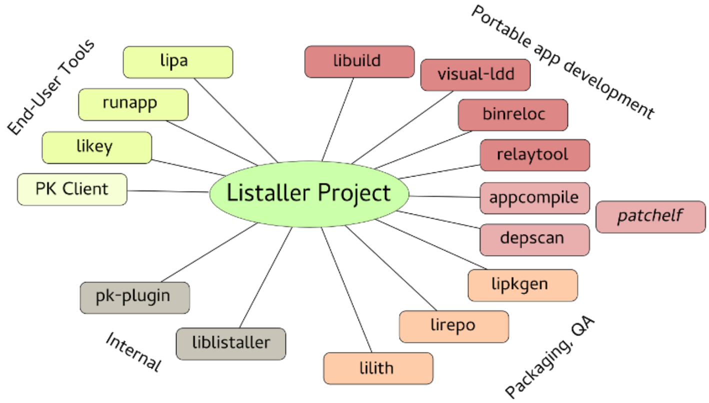


Basic concept

- Listaller contains a PackageKit plugin, mediating between Listaller and PackageKit
- The plugin acts as »meta-backend«, sending information about Listaller packages via PackageKit's DBus interface, and making queries to the native backend
    - Every PackageKit client can install, remove and update Listaller packages
- Listaller installs some XML for AppStream-compatible software centers, so they can display details about a 3rd-party application
- Uses a superset of the AppData specification as source for application metadata



Listaller Tools



Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The depscan tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple LZMA2-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple LZMA2-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

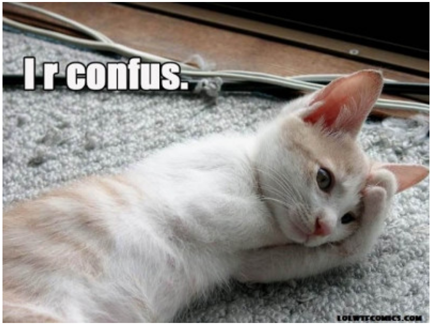
- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple LZMA2-compressed tarballs

Utopia package creation

1. Write AppStream AppData describing the application
2. Write a small `pkoptions` file, defining few basic options for the package
3. Write a file/dir listing for the new package
4. Build the package!

- Listaller will use `appcompile` to determine the buildsystem and build the app, install it to a temporary location and then add it to the package
- The `depscan` tool is used to determine dependencies and map them to a component (e.g. libglib2 is part of the GLib2 component)
- Package is GPG-signed afterwards
- During installation, a few optimizations and adjustments are made automatically
- A Listaller IPK installation does not run any user-defined script
- Listaller packages are simple `LZMA2`-compressed tarballs

Dependency solving?



Dependency solving...

Problem #1

Satisfying dependencies is a complex task, searching for the right dependency is difficult and takes time. We want to avoid complex dependency-solving.

Problem #2

What if the distributor does not provide enough metadata to find a dependency?

Problem #3

What if the ABI of a dependency is not stable? What if we absolutely require an old library version?

Dependency solving...

Problem #1

Satisfying dependencies is a complex task, searching for the right dependency is difficult and takes time. We want to avoid complex dependency-solving.

Problem #2

What if the distributor does not provide enough metadata to find a dependency?

Problem #3

What if the ABI of a dependency is not stable? What if we absolutely require an old library version?

Dependency solving...

Problem #1

Satisfying dependencies is a complex task, searching for the right dependency is difficult and takes time. We want to avoid complex dependency-solving.

Problem #2

What if the distributor does not provide enough metadata to find a dependency?

Problem #3

What if the ABI of a dependency is not stable? What if we absolutely require an old library version?

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Components to the rescue!

- Most 3rd-party applications don't need complex dependency-solving, they are built against a certain development platform
  - Apps might require GTK+3 (>= 3.12), or Qt5 (>= 5.2) or PulseAudio (>= 2.4)
- A component is e.g. a toolkit, a basic building block the systems consists of. Public interfaces a component provides are described in a component-definition file.
- Listaller only needs to ensure that basic components are present for the application to run. Minor dependencies can be shipped with the application package
- Upstream should write component-definition files and ship them with their source
- Some dependencies can be satisfied by downloading a statically linked version provided by upstream or the distributor, or querying cpan/cran/pypi/gems/...

Component Information

- Think of it as an »non-developer pkg-config file«
- Describes the component version and all public interfaces this component provides
- Can map software dependencies to a component at package-build-time and a component to a native-package at install time

```
# GLib C utility library
ID: GLib2
Name: GLib 2.0
Version: 2.36
VersionDynamic: shell$ glib-fake --version
Libraries: libgio-2.0.so.0
 libgobject-2.0.so.0
 libglib-2.0.so.0
Binaries:
 gdbus
 gio-querymodules
 glib-compile-resources
 glib-compile-schemas
 gresource
 gsettings
```

Component Information

- Think of it as an »non-developer pkg-config file«
- Describes the component version and all public interfaces this component provides
- Can map software dependencies to a component at package-build-time and a component to a native-package at install time

```
# GLib C utility library
ID: GLib2
Name: GLib 2.0
Version: 2.36
VersionDynamic: shell$ glib-fake --version
Libraries: libgio-2.0.so.0
 libgobject-2.0.so.0
 libglib-2.0.so.0
Binaries:
 gdbus
 gio-querymodules
 glib-compile-resources
 glib-compile-schemas
 gresource
 gsettings
```

Component Information

- Think of it as an »non-developer pkg-config file«
- Describes the component version and all public interfaces this component provides
- Can map software dependencies to a component at package-build-time and a component to a native-package at install time

```
# GLib C utility library
ID: GLib2
Name: GLib 2.0
Version: 2.36
VersionDynamic: shell$ glib-fake --version
Libraries: libgio-2.0.so.0
 libgobject-2.0.so.0
 libglib-2.0.so.0
Binaries:
 gdbus
 gio-querymodules
 glib-compile-resources
 glib-compile-schemas
 gresource
 gsettings
```

Component Information

- Component definitions are shipped with every Listaller (IPK) package, they are available on every distribution
  - Data provided by the distributor overrides data shipped with the package
- Component info can contain commands to extract a version number to determine the current component version on a new distribution
- In future, component data might refer to a statically linked copy of the component, to satisfy dependencies of old software
- The Listaller helper tool `depscan` is able to automatically scan the software binaries and match them to components

Component Information

- Component definitions are shipped with every Listaller (IPK) package, they are available on every distribution
    - Data provided by the distributor overrides data shipped with the package
- Component info can contain commands to extract a version number to determine the current component version on a new distribution
- In future, component data might refer to a statically linked copy of the component, to satisfy dependencies of old software
- The Listaller helper tool `depscan` is able to automatically scan the software binaries and match them to components

Component Information

- Component definitions are shipped with every Listaller (IPK) package, they are available on every distribution
  - Data provided by the distributor overrides data shipped with the package
- Component info can contain commands to extract a version number to determine the current component version on a new distribution
- In future, component data might refer to a statically linked copy of the component, to satisfy dependencies of old software
- The Listaller helper tool `depscan` is able to automatically scan the software binaries and match them to components

Component Information

- Component definitions are shipped with every Listaller (IPK) package, they are available on every distribution
  - Data provided by the distributor overrides data shipped with the package
- Component info can contain commands to extract a version number to determine the current component version on a new distribution
- In future, component data might refer to a statically linked copy of the component, to satisfy dependencies of old software
- The Listaller helper tool `depscan` is able to automatically scan the software binaries and match them to components

Component Information

- Component definitions are shipped with every Listaller (IPK) package, they are available on every distribution
  - Data provided by the distributor overrides data shipped with the package
- Component info can contain commands to extract a version number to determine the current component version on a new distribution
- In future, component data might refer to a statically linked copy of the component, to satisfy dependencies of old software
- The Listaller helper tool `depscan` is able to automatically scan the software binaries and match them to components

Future

- Finalize and formalize Listaller specifications
- Get upstream projects to ship component-definitions by default
- Get Listaller into more distributions
- Maybe in the end have a way for 3rd-party developers to target all Linux distributions with one app-package?

Future

- Finalize and formalize Listaller specifications
- Get upstream projects to ship component-definitions by default
- Get Listaller into more distributions
- Maybe in the end have a way for 3rd-party developers to target all Linux distributions with one app-package?

Future

- Finalize and formalize Listaller specifications
- Get upstream projects to ship component-definitions by default
- Get Listaller into more distributions
- Maybe in the end have a way for 3rd-party developers to target all Linux distributions with one app-package?

Future

- Finalize and formalize Listaller specifications
- Get upstream projects to ship component-definitions by default
- Get Listaller into more distributions
- Maybe in the end have a way for 3rd-party developers to target all Linux distributions with one app-package?

Conclusion I

- We need a solution to make it easy for 3rd-party developers to distribute their software on all distributions at once. We also need to increase visibility of existing applications in distributors repositories
- AppStream provides all metadata you want to build a software center
- The libappstream library abstracts all remaining differences, and is an implementation of AppStream which can be shared by all clients who want to make use of it's features
- Listaller provides a way to ship applications on all Linux distributions and keep them up-to-date, reusing existing user interfaces
- The Listaller project is not yet mature and the specifications are not finalized, but this will happen soon with the 0.6 release

Conclusion I

- We need a solution to make it easy for 3rd-party developers to distribute their software on all distributions at once. We also need to increase visibility of existing applications in distributors repositories
- AppStream provides all metadata you want to build a software center
- The libappstream library abstracts all remaining differences, and is an implementation of AppStream which can be shared by all clients who want to make use of it's features
- Listaller provides a way to ship applications on all Linux distributions and keep them up-to-date, reusing existing user interfaces
- The Listaller project is not yet mature and the specifications are not finalized, but this will happen soon with the 0.6 release

Conclusion I

- We need a solution to make it easy for 3rd-party developers to distribute their software on all distributions at once. We also need to increase visibility of existing applications in distributors repositories
- AppStream provides all metadata you want to build a software center
- The libappstream library abstracts all remaining differences, and is an implementation of AppStream which can be shared by all clients who want to make use of it's features
- Listaller provides a way to ship applications on all Linux distributions and keep them up-to-date, reusing existing user interfaces
- The Listaller project is not yet mature and the specifications are not finalized, but this will happen soon with the 0.6 release

Conclusion I

- We need a solution to make it easy for 3rd-party developers to distribute their software on all distributions at once. We also need to increase visibility of existing applications in distributors repositories
- AppStream provides all metadata you want to build a software center
- The libappstream library abstracts all remaining differences, and is an implementation of AppStream which can be shared by all clients who want to make use of it's features
- Listaller provides a way to ship applications on all Linux distributions and keep them up-to-date, reusing existing user interfaces
- The Listaller project is not yet mature and the specifications are not finalized, but this will happen soon with the 0.6 release

Conclusion I

- We need a solution to make it easy for 3rd-party developers to distribute their software on all distributions at once. We also need to increase visibility of existing applications in distributors repositories
- AppStream provides all metadata you want to build a software center
- The libappstream library abstracts all remaining differences, and is an implementation of AppStream which can be shared by all clients who want to make use of it's features
- Listaller provides a way to ship applications on all Linux distributions and keep them up-to-date, reusing existing user interfaces
- The Listaller project is not yet mature and the specifications are not finalized, but this will happen soon with the 0.6 release

Conclusion I

- We welcome contributors to AppStream! Feedback is always wanted, and we are waiting for software-center implementations!
- Listaller needs developers as well! If you have an idea or want to improve something, get in contact!

Conclusion I

- We welcome contributors to AppStream! Feedback is always wanted, and we are waiting for software-center implementations!
- Listaller needs developers as well! If you have an idea or want to improve something, get in contact!

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Conclusion II

- Writing Freedesktop standards is hard
- Never do things leading to the impression that you develop the standard primarily for GNOME/KDE
- Standards are created by implementing something, not by calling it a standard
- Communicate and blog as much as possible while introducing a new thing, so people know about it and can give feedback
- Sometimes it needs someone to just go ahead with a project and have others follow
- Discuss things with a small amount of people first, then open up to the wider community - ignore people bikeshedding about details, always ask for a better proposal

Thank you for your attention!

(Further) Questions?

Useful links

PackageKit: http://www.packagekit.org/

Listaller: http://listaller.tenstral.net/

AppStream Documentation:
http://www.freedesktop.org/software/appstream/docs/

AppData Information & Validation:
http://people.freedesktop.org/~hughsient/appdata/