kyla Documentation

Release 2.0.1

Matthäus G. Chajdas

Dec 24, 2017

Contents

Contents:

CHAPTER 1

Quick start

The fastest – and also the recommended way – to get started with kyla is to use the Python bindings to create a build definition. A simple installer could look like this:

```python
import kyla

rb = kyla.RepositoryBuilder ()

mainFeature = rb.AddFeature ()

binaryFiles = rb.AddFileGroup ()
binaryFiles.AddDirectory ('path/to/bin', outputDirectory = 'bin')

# We want to store the binary files in their own package
binPackage = rb.AddFilePackage ('bin')

# Link the binary package to the binary files
binPackage.AddReference (binaryFiles)

# Our main feature consists of only the binary files -- link them together
mainFeature.AddReference (binaryFiles)

# For the UI, we want a single node feature tree
featureTreeMainNode = rb.AddFeatureTreeNode ('Binaries',
    'These are the binaries')

# link the feature tree node to the feature. If the user selects the node,
# all linked features will be installed
featureTreeMainNode.AddReference (mainFeature)

open ('desc.xml', 'w').write (rb.Finalize())
```

The Python bindings hide the complexity of referencing nodes and handle all ids internally. The generated Xml file can be then processed using `kcl` to create a build package using the following command line: `kcl build desc.xml target-folder`.

Once created, there are two ways to install. The command line can be used as following:

```
$ kcl query-repository features path/to/repository
c353d049-710e-4027-a707-18e11bbcab22

$ kcl install path/to/repository path/to/target c353d049-710e-4027-a707-18e11bbcab22
```

Note that the feature id will vary, this is just an example. The alternative is to build the user interface and create an `info.json` file next to the binary with the following contents:

```
{
    "applicationName": "Name to show in the UI",
    "repository": "path/to/repository/relative/to/UI"
}
```

CHAPTER 2

How do I ...?

How to update an installation

Updating is performed through a *configure*. Let's assume you have two version of your product, v1 and v2. The customer has installed v1 of the product. An upgrade can be performed by a configure using the installer of v2 directly into the current installation directory. The only difficulty arises if the installation has multiple features, in which case you need to mape the currently installed features to the features in v2. As there's no general solution for this (one feature could be split, for instance), kyla does not handle this directly. Instead, you need to query the currently installed features in the target directory, and map them manually to the new features in the v2 repository.

Even though for kyla, v1 and v2 are unrelated repositories, kyla will only transfer new file contents from v2. There is no need for a special patch installer.

How to add/remove features

Adding/removing features is done through *configure*. Note that the current target directory cannot be used as both a source and destination of a configure operation, even though removing features does not strictly require the source to be present.

How to do a web installation

Create a packed installation repository, and host it on an HTTP server with range requests enabled. Use the full URL including `http://` or `https://` when opening the installation source repository, and kyla will automatically download the repository description.

CHAPTER 3

Repository build description

Installer repositories are compiled by using `kcl build`. This requires an Xml file that describes how to build a file repository. Here's an example repository file:

```xml
<?xml version="1.0" ?>
<Repository>
  <Features>
    <Feature Id="7f9e0d24-1c82-49d3-86f9-47c0a7c984d5">
      <Reference Id="c9a41343-bc90-4871-84bd-27f04aac4794"/>
    </Feature>
  </Features>
  <Files>
    <Group Id="c9a41343-bc90-4871-84bd-27f04aac4794">
      <File Source="A:\full\path\file.exe" Target="bin/file.exe"/>
    </Group>
    <Packages>
      <Package Name="bin">
        <Reference Id="c9a41343-bc90-4871-84bd-27f04aac4794"/>
      </Package>
    </Packages>
  </Files>
  <UI>
    <FeatureTree>
      <Node Description="Binary files." Name="Binaries">
        <Reference Id="7f9e0d24-1c82-49d3-86f9-47c0a7c984d5"/>
      </Node>
    </FeatureTree>
  </UI>
</Repository>
```

Reference

The repository stores various objects which can be referenced. In general, anything which has an `Id` attribute can be referenced, and all references are done by adding a `Reference` node. Objects can be grouped by using a `Group` node, this makes it possible to reference many objects using a single `Reference`.

- `Repository` is the top level node and must be always present.
- `Features` contains the feature list. A feature must reference another object in the repository, and must not reference another feature.
- `Files` describes all file objects and the storage layout.

  If present, `Packages` is used to group files into packages. A `Package` must have a name and it must reference an object from the `Files` tree. Files which are not explicitly packaged are automatically placed into a `main` package.

  Files can be grouped together for easy referencing using a `Group` node.

  A `File` node can reference the full source path or a relative path. If a relative path is used, the source directory must be specified during the compilation. Relative paths are automatically used for the `Target` path as well if there's no `Target` specified.
- `UI` contains UI related data. This data is only present in source packages and never deployed, but it's used to create the installation UI.

  The `FeatureTree` defines the feature grouping. At least one `Node` must be present, and every node must reference at least one `Feature`. Nodes can be arbitrarily nested.

CHAPTER 4

Repository types

kyla supports several *repository types* with different capabilities. The built-in types are:

- `Deployed`: A deployed repository is the "installed" state, that is, the content objects are stored with their actual file name, and some content objects may be duplicated. A deployed repository supports repair, add/remove, and validation.
- `Packed`: A packed repository consists of the database and one or more package files. Content objects are spread over package files. A packed repository supports only validation.

A `Packed` repository can be also be used for web installation. Putting all files onto a server which supports HTTP range requests makes the packed repository readable over the web.

Supported operations

Type	Installation source	Verify	Repair	Configure
Deployed	Yes	Yes	Yes	Yes
Packed	Yes	Yes	No	No

CHAPTER 5

Comparison with other tools

NSIS

The Nullsoft Scriptable Install System is a popular and fully fledged installation system. It works at a much higher level than kyla and supports the creation of shortcuts, has a scripting language, and creates self-extracting installations.

The main differences are:

- kyla has native support for updating and updates only the required files - NSIS does not have built-in support for this.
- NSIS is not designed for integration - it takes care of dialogs, for example - while kyla is.
- NSIS does not support repairing or changing installations.
- NSIS supports only Windows (the generated installers only support Windows)[1].

InnoSetup

InnoSetup is another popular Windows installation system. Similar to NSIS, it provides a high-level installation framework, and comes with it's own scripting engine and even an IDE.

The main differences are:

- kyla has native support for updating and updates only the required files - InnoSetup can update, but it is not aware which files have changed and which haven't.
- InnoSetup is not designed for integration - it takes care of dialogs, for example - while kyla is.
- InnoSetup does not support repairing or changing installations.
- InnoSetup supports only Windows.

[1] From http://nsis.sourceforge.net/Features: Generated installer will still run on Windows only

Windows Installer

Windows Installer is also a fully fledged runtime for installations which is bundled with Windows. It is comparable with kyla as it also uses a database, but there are again many differences:

- kyla can install directly from web, while the Windows Installer will download a package first before it starts installing[2].
- Windows Installer does not support modern compression algorithms. Only cabinet files are supported.
- Windows Installer cannot split a file when creating an installation media - very large files will result in very large packages.[3].
- Windows Installer supports only Windows.

[2] From https://msdn.microsoft.com/en-us/library/windows/desktop/aa368328(v=vs.85).aspx: If the installation database is at a URL, the installer downloads the database to a cache location before starting the installation.

[3] From http://www.joyofsetup.com/2011/06/21/wix-and-cabinetry/: On the flip side, any individual files that are larger than the maximum size go into a single-file cabinet, so it's possible that such cabinets will be larger than the maximum size.

CHAPTER 6

Changelog

kyla 2.0.3

- Progress update has been rewritten and is now much more accurate. Deleting files reports progress now so removing features shows progress, and retrieving files reports progressed based on data written, including file duplication.
- Fix memory leak if a repository could not be opened.
- UI has been simplified and uses the "fusion" style for cross-platform portability.
- Updated SQLite from 3.19.2 to 3.21.0.

kyla 2.0.2

- Installation preparation time has been drastically cut down if many (>10000) files are to be installed. Previously, a lot of time was spent identifying unique paths to create all directories, which has been reduced by more than an order of magnitude.
- Progress reporting is now solely based on fetch progress. This optimizes for the default case where progress is dominated by fetching changed contents. In that case, the progress will be 100% accurate and reflect exactly what is written on disk. In cases where an installation mostly consists of duplication, the progress will not be updated while files are duplicated. Renames are treated as new content fetching.

kyla 2.0.1

- The packed repository deployment backend is now multithreaded. It will overlap reading data, decompressing, and writing data. For network sources, this means that downloads will happen concurrently with everything else, and installations should finish as soon as the data download has finished.
- On Windows, the sample UI now shows the progress in the task bar.

- Updated SQLite from 3.16.1 to 3.19.2.

kyla 2.0

Warning: This release contains a breaking change of the file format. Kyla 1.0 will fail to open a package created with Kyla 2.0 and vice versa.

- Changed database format for improved forward compatibility.
- kyla uses now a feature-based definition instead of the previous file-set centric approach. This changes both the database structure, as well as the repository definition structure.
- The Python generator has been rewritten.
- The `kui` sample UI has been rewritten, and kyla learned how to provide UI related information.
- Zstd compression is available as an alternative to the default Brotli compression.
- The `KYLA_MAKE_API_VERSION` macro has been fixed.
- The repository builder can print out various statistics now, for instance, the final compression ratio.
- kyla supports file content encryption.

  Note: The encryption only encrypts the file contents, so file names, sizes, and even the hashes of the content will remain visibile. In particular, the database itself is not encrypted.

CHAPTER 7

Building kyla

Requirements

- A modern C++ compiler: On Windows, Visual Studio 2015 or later, on Linux, GCC 6 or later
- CMake 3.8 or later
- Boost 1.63 or later. The `filesystem`, `program_options`, and `system` components must be available.
- OpenSSL 1.1 or later
- Python 3.5 or later
- Qt 5.6 or later. Qt 5.9 is recommended. kyla can be optionally built without the UI, in which case Qt is not required.

Note: To build kyla without the UI, disable the `KYLA_BUILD_UI` option.

Build

Run CMake to configure the project. Build the `kcl` target to get the kyla runtime.

Note: The `docs` target currently supports Windows only due to the virtual environment setup.

Tests

kyla comes with a set of sanity tests which ensure basic functionality. To run the tests, build the `RUN_TESTS` target, or use `CTest`.

CHAPTER 8

Licenses

Acknowledgements

- Andreas Weis (der_ghulbus@ghulbus-inc.de)
- Raymund Fülöp (r@ymund.de)

Third-party licenses

kyla uses third-party code. The licenses for that code are as following:

Brotli

LLVM

Parts of this software are based on LLVM.

OpenSSL

OpenSSL License

Original SSLeay License

pugixml

zlib

The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files http://www.ietf.org/rfc/rfc1950.txt (zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format).

Zstd

CHAPTER 9

Introduction

kyla is an installation system designed to deploy large amounts data. The main features are:

- Speed: kyla uses fast compression algorithms, reads and writes files sequentially, and only fetches the data it absolutely needs.
- Scalability: Tens of thousands of files can be deployed easily. kyla also has first-class support for large binary files, splitting them up as necessary to provide streaming installations.
- Web-first: kyla can install directly from the internet, fetches only the minimum amount of data required, and installs during download to maximize bandwidth usage and minimize installation time.
- Feature-based installation: Deploy only subsets of your application, and support configure functionality.
- Upgrades, downgrades, configure: kyla can upgrade/downgrade your installation to another version and will only fetch changed contents. Upgrades, downgrades and configurations are handled through the same function.
- Library design: kyla is designed to be embedded into your frontend. It provides an easy-to-use C API and can be statically or dynamically linked.
- Reliability: It uses the SQLite storage engine for all metadata storage - one of the most robust databases in the world. Installations can be validated and repaired if they ever get corrupted.

Note: kyla is not a full-fledged installer taking care of registry keys, registering services, or similar. It is designed to deploy and manage applications in a single folder. If you need additional pre/post install hooks, you can easily build them on top of kyla. For a comparison with existing tools, check out the *Comparison with other tools*.

CHAPTER 10

System requirements

kyla has been tested on the following OS:

- Windows 10 x64
- Ubuntu 17.04

Other Windows/Linux variants should work, as kyla only relies on few cross-platform libraries, but they're not tested regularly.