# Jade:DS Integration Guide

## How Jade:DS works

### About Jade:DS

Jade:DS is a complex solution to distribute and manage software products, especially computer game and similar. It is provided as a service by Mediaguild UG and allows developers to self-publish their products. Build upon the system, additional third party services are available, such as the little-indie portal ran by z-software.

The main focus of Jade:DS are independent game developers, who aim to publish their games by themselves. The developer has a lot of freedom in design and process, but can always rely on the high integrated backend. The optional sharing of products and privileges allows the cooperation between developers to reduce effort and to increase range.

Jade:DS can be used as a non-exclusive platform, but optimizes its benefit from being a central platform for customer relationship, community management and sales. Developing exklusive features for this platform enables a game to use a much higher potential with a minimum of efforts, such as the tight integration between the systems online storage and the CMS. This allows the visual representation of the progress of the player within the game on the web pages, by providing additional content, game help or even trans-game benefits.

The possiblities of Jade:DS are limitless, and if you have ideas about features, which aren't available yet, please contact us. We're always open to extend the system and give it the little extra plus to make your game shine a bit more than elsewhere!

### The Components

Jade:DS consists of three main components, a content management system for the web browser, a server facility and a client, which needs to be installed on the customers computer. These three components are connected and provide the final services. The system can be parallelized in most of its components. This allows us to scale the system in short terms of time to the present needs, which optimizes costs and availability.

All three main components are fed in a direct or indirect way by our backend database. This database stores all system and customer information and makes the tight integration possible.

Products are organized in archives. These archives are snapshots of your project and are build locally on your development machine. Updates are new revisions of these archives, which contain only the changed files and the new directory layout. These archives are uploaded to the server, which distributes them to the running file servers. These file servers deploy chunks of data to the customers client, depending on the requested revision. The caches of the locale client are able to even hold more than one valid revision at once. The game can access non-executable files directly from these caches, which saves disk space and allows the game to remain playable even throughout an update.

### Checklist: Integrate an existing Project

The following checklist is a brief overview of the integration process, as well as a point by point list to assist you in the real process. The process describes, how to implement and test a project with the live-system client. This process will be replaced later with a standalone simulator, which allows local building and testing before a self-contained upload. This checklist is available as a seperate document, too. You can use that to print yourself a personal copy.

- ☐ Let the product be registered by the Jade:DS team
- ☐ Build and upload a valid declaration archive without referenced archives
- ☐ Activate the declaration archive and perform a manuall buy
- ☐ Run the client to implement status features
- ☐ Build and upload the asset archives
- ☐ Implement file access features

- ❑ Add a valid setup script to your binary archive
- ❑ Build and upload the binary archive
- ❑ Build and upload a declaration archive with referenced archives
- ❑ Run the client to test

Please keep in mind, that most steps requite actions by the Jade:DS team at the moment. It is recommended to keep contact during the integration project with us to ensure a flawless process.

## Building Archives

To build the archives, used by the Jade:DS servers, we provide a tool named wxArchive. You can find this tool within the bin-directory, as well as in your start menu. This tool delivers a comprehensive online help. We'll cover only the general concept here.

**Important:** The archives are supporting ascii encoding for filenames only at the moment. Be sure, your filenames only use a portable naming convention.

**Important:** The archives can't store empty directories, as they store files and directories are only a virtual concept to them.

## The declaration Archive

The declaration archive is an archive, which is downloaded to the customers client as soon as he bought or unlocked the corresponding product. It contains the title, preview thumbnails and the assigned archives. The items, which are used to preview the product in the client can be localized for multiple languages. The main file is the `product.xml`, a plain xml file. All other files in this archive need to be referenced by the xml first. At the moment, this can only be png files, which can be used as thumbnails. The wxArchive utility contains a small validator and simulator for declaration archives. Be sure to use this tool before building a new revision.

**Important:** Although the file contains a reference to the executable file, no executable files are ment to be part of the declaration archive. It is only possible to launch a product, if all referenced archives are downloaded, so it is not necessary for the executables to be in there, nor is it wanted.

The most outer tag pair in the `product.xml` - this file name is mandatory - is the `<product>`-tag. It needs an attribute `encryption`, which is reserved for future encrytion methods, but yet the attribute needs to be set to none. A second mandatory attribute is `default`, which defines the default language of the game and the views.

### View-Tags

Within the `<product>`-tag, several `<view>`-tags can be defined, one for each available language. Each `<view>`-tag needs a language attribute, which can be one of the system languages. You should specify the default attribute set to `true` to the most common language, in general, this would be english.

| **Language-ID** | **Language** |
|---|---|
| en | English |
| de | German |

A `<view>`-tag consists of a handful further tags, which are all mandatory. The first one is the products title, the `<title>`. This is the decorative title, which is displayed in the clients product list. The title is embedded between the opening and closing tag. The second is the `<subline>`-tag, which can be a further description of the product. Finally, there is the `<preview>`-tag, which contains the default thumbnail, marked in the `<primary>`-tag, and optional a set of further thumbnails, marked in one or more `<rollover>`-tags. Both, `<primary>` and `<rollover>`, expect the attribute image with the filename of the thumbnail. Each `<rollover>`-tag needs the attributes duration and effect, where duration specifies the display time in milliseconds, before it transists to the next image and effect specifies the transition effect. Currently, this is limited to the fade effect, e.g. `effect = 'fade:50'`, where 50 is the duration of the fade in milliseconds. Currently, the images are not scaled and need to measure 88 by 58 pixels.

## Product-Tags

Optionally, you can provide a product icon, which is sized 32 by 32 pixels. This icon is used in contexts, where the product is referenced in a message, e.g., if the download or update of your product is finished. If this icon is not provided, a generic one is used instead.  Similar to other image resources, the attribute `image` is used for the file name.

A mandatory  tag within the `<product>`-block is the `<launch>`-tag. It defines the executable associated with this product in the `bin`-attribute and the working directory in the `path`-attribute. The `platform`-attribute defines the platform the executable runs on. There must be one `<launch>`-tag per platform in this block.

| **Platform-ID** | **Platform** |
|---|---|
| generic | Cross platform archive |
| win32 | MS-Windows |

Finally, we define the associated archives in the `<product>`-block. Each `<archive>`-tag defines one archive and requires a `platform`-attribute and an unique `name`-attribute. By default, these archives are not visible to the application, as the `nomount`-attribute is set to `true`. To enable visibility, you need to provide the `mount`-attribute with a path, which preceeds the paths within the archive.

## Archives with Setup

If an archive needs to install files in the local file system, e.g., because there is no way to access it from the archive, like binaries, it needs a file called `setup.xml`, which defines the installation process. The main-tag for this file type is `<setup>`. This block is divided into three main sections, one declared in the `<locales>`, one in the `<directories>` and one in the `<temporary>`-tag.

The setup is called upon the first start of the application, after the setup was received initially or has been updated. The Jade:DS client verifies the setup script for changes, e.g. if the script has been modified itself, or any of the referenced binary file. The setup script is executed completly each time such a change happend.

## Locales-Tags

The locales block allows you to declare labels for several languages, which are used to display the progress of the installtion. This allows you to give more detail information during the setup as well as to group several files to one label. This is very useful, if you have a lot of smaller files, which are copied that fast, that reading them wouldn't provide any information. A single label provides a clearer view on the installation here.

The language used to display the labels is the langauge of the client and if that is not provided, the default language is used. The labels are used instead of file names, as referenced with the `alias` attribute in all tags of the setup. The `locales` tag expects a valid `default` attribute, which holds the language to be used by default. The labels are organized in `label` tags, where an `id` attribute defines the label for reference. In this block, you can place one or more `string` tags. These tags are assigned with a `language` attribute to a specific language. The text to be displayed is placed within the `string` tag.

## Directories-Tags

This block defines the installation process itself. The most important tag is `copy`. The `file` attribute specifies the requested file from the archives and it will be copied to the current directory. The base directory is the product directory under bin and can't be altered by name. Within the block, you can access directores, which are created and named by the `name` attribute of the subsequent `directory` block. The directoy structure has to be identical with the one within the archive, if you want to copy files. You can create empty directories freely, too.

Specific to the windows platform, there is another tag called `win32:link`. This is used to provide a link in the start menu to a file within the current directory. The `name` attribute defines the visible name in the start menu. The `target` attribute names the executable file to which the link refers. The link can have a description associated, which is defined by the `description` attribute.

### Temporary-Tags

The temporary block is very similar to the directories block, but it can't contain further directories, nor can it install files to a client location. Instead, the files are copied into the users temporary directory. The `execute` tag provides the same functionality as the `copy` tag, but executes that file as well. You can provide additional arguments with the `arguments` attribute, e.g. to surpress a GUI of an external installer.

Within the temporary block, you can remove the copied or executed file with the `remove` tag. You need to do this manually to make sure there are no files left, which polute the customers hard disk.

## Managing Archive Revisions in the System

All data, which is deployed by Jade:DS, is organized in archives. These archives are built with the wxArchive utility available with this SDK. The utility comes with an extensive online help, so we concentrate on the general process here. Archives are organized in revisions. Each time you want to deploy a new version of one or more archives, you create a new revision and upload it to your server space. Placed within the `archives` subdirectory, you can populate it with the web interface to the customers.

Archives are build for a specific platform or as generic archives. This makes it easier to maintain multiplatform titles and divides installable archives from archives, which are accessed through the client.

| **Platform** | **Description** | **Naming recommendation** |
|---|---|---|
| Generic | Suitable for all platforms, most likely used for assets, like graphics and sound | productname<br>productname_assets |
| Win32 | Containes binaries for x86 windows | productname_win32binaries |

## Using the Online-Storage

### StorageSlots

StorageSlots are the core mechanism of all online storage features covered by Jade:DS. They are data fields, which are assigned to a customer and a product. Each slot has a data type and an operator.

| **Type** | **Description** | **Default value** |
|---|---|---|
| String | A plain string, which can be displayed by standard output routines | Empty string |
| Numeric | An integer field | Zero |
| Time | A time field | Zero |
| Binary | Binary data, which shall not be displayed as text. | Empty string |
| Boolean | True or False | False |

| **Operator** | **Behavior** | **Response** |
|---|---|---|
| Set | The slot is set to the provided value | None |

| | | |
|---|---|---|
| Increment | Increments a numeric slot by a numeric value | New slot value |
| Max | Sets a numeric or time slot to a numeric value, if the value is greater than the slot | New slot value, if changed |
| Min | Sets a numeric or time slot to a numeric value, if the value is lower then the slot | New slot value, if changed |

### StorageViews

A storage view is an ordered table from several slots. The most common use for storage views are highscore lists.

### Achievements

Achievements are build on storage slots by connecting them with conditions. An achievement defines a set of conditions and requires a number of them to be unlocked. An achievement can require further achievements, too, which implements meta-achievements.

## Managing the Client Status

The connection to the client should be managed within your main loop. There, you should notify the runtime pump to inform the client about the time status and query the pending status objects. Don't try to do this in a synchronous way, as it is not guaranteed, that you'll only receive the message objects you requested.

## Wrappers

We try to provide complete wrappers for major engines and SDKs, if possible. Currently, we supply an out-of-the-box solution for Ogre. Please contact us for your needs regarding other third party products, we may either provide you with a solution or can assist you in creating a suitable one.

### The OGRE Wrapper

The OGRE wrapper comes in two flavors, one to be statically linked and one as a plugin dll. Depending on your framework, one or the other is the best solution for you, but the plugin is the most versatile one. To use it, you simply need to copy the dll into the binary directory of your game and modify the `plugin.cfg` file. You can replace the `FileSystem` or `Zip` source with `JadeDS` and the plugin automatically retrieves the assets from the locale caches. The only thing you need to do before is to connect to the client, because the SDK needs to know how to mount to the caches.

The notifyTick and the updateStateSync calls can be placed safely in the `frameRenderingQueued` callback. Using either the `stateSyncListener` or polling the state objects manually works both equally well with OGRE.

## Contact

In case of questions, requests or failures, please contact us:

Mediaguild UG

Weststrasse 4

42555 Velbert

Tel.: +49-2052-926778

eMail: Developer@Mediaguild.de

# Glossary

In the following, we collected some catch words, you'll always read and hear during the process of integration. To avoid confusion, we tried to find clear definitions for this vocabulary. If you believe, there is something missing here, please drop us a note, most likely it will be added for the next release of the SDK.

| | |
|---|---|
| ***Achievement*** | An achievement is a meta scoring instrument, used to motivate the player and track his progress. Usually, unlocking an achievement is displayed immediately above the UI panel or in the upper-right corner of the screen. Achievements can track the players gaming quality and progress as well as adding some sense of humor by pushing the player to try funny things even during a seriouse game. |
| ***Archive*** | An archive is a file, which packages several files and directories in a preprocessed structure, which can be used by Jade:DS. They are the origin of all file access, but they are used only by the packaging developer and on the servers. They are transfered partially on request to the customers client, where these chunks of data reside in local caches. An archive can have multiple revisions, where each revision is represented by a single file with similar naming and different revision numbers. These revision files are connected internally and frame up to the archive itself. |
| ***Assets*** | Beside its executable, a game consists of a lot of different types of files, e.g. graphics, sounds and meta-data files. These files are called assets, as they make the value of a game. These files are generally stored in archives, if intended to be used in a Jade:DS product. |
| ***Cache*** | The customer client uses cache files to store the chunks of data received from the products archives. A cache can contain one or even more valid revision states. This allows to precache a new revision of a game and keeping the game in a runnable state, by preserving the previouse revision. |
| ***Client*** | Each customer needs to install a local client software. This client handles all synchronization of the products data and the communication with the backend servers, e.g. for license validation, online storage and player to player communication. The client is designed to be very lightweight in processor time and memory usage. |
| ***Declaration*** | The client needs to be fed with several flields of meta data about a product. This is stored in a declaration archive and provides localized strings for title and subline, a preview thumbnail and an optional slide show of thumb images and the information needed to mount the products archives and to launch the product. |
| ***Meta-Achievement*** | A meta-achievement is an achievement, which isn't unlocked by a condition on a storage slot, but by unlocking one or more other achievements. |
| ***OGRE*** | OGRE stands for open graphics rendering engine and is a 3d rendering engine, developed by a very active open source team. It is used in several commercial titles and this SDK provides special wrappers for this engine and its very good resource system. |
| ***Product*** | A product within Jade:DS is a buyable entity, which defines a specific game and its assets, both localized within the archives of the product. The archives are assigned to the product by the declaration archive. |
| ***Revision*** | A revision is a specific state of an archive. It holds all new or modified files compared to its parent revision - or simply all files in case of the first revision. A revision can be numbered freely, either number by number or using the revision number of your source control system. As long as a new revision has a higher number than its parent, everything is fine. A revision is a single file, as one or more of these revision files frame an archive. |

|  |  |
|---|---|
| ***StorageSlot*** | StorageSlots are server-sided data containers, which store persistent data for the the game. They can be of different data types and numeric values can be modified by certain operators. Storage slots are the base data for storage views and achievements. |
| ***StorageView*** | While a storage slot contains a single value for a specific customer, a view accumulates several slots and relates it to the values of other customers. The common use of this are highscore lists. |
| ***Wrapper*** | A wrapper is an interface, which covers another interface by adding or altering its functionality. Wrappers are used to add the Jade:DS API to other languages or engines, like the OGRE wrapper, which provides generic file access to Jade:DS archives, using the built-in resource concept. |
| ***wxArchive*** | wxArchive is a utility delivered with the Jade:DS SDK. It is used to build revisions of archives and upload them to the server. It is a simple tool, but provides validation utilities for the specialized files, like the `setup.xml` or the `product.xml`. |