PlutoF

QR codes in the PlutoF ecosystem

User tutorial

Document information

Date and version no.	Author(s)	Comments/Changes
30.03.2026, v0.1	Filipp Ivanov	Initial draft.
31.03.2026, v0.2	Allan Zirk, Kessy Abarenkov	Reviewing the text and applying document styling.

Table of Contents

Introduction

This tutorial provides an overview of QR code specifics and their functionality in PlutoF.

Terms

Link-based QR code – a QR code that contains a PlutoF internal ID pointing to a specific record.

Entry QR code – a QR code used for entering data. It does not include a PlutoF record ID, but it contains enough information for entry validation and for resolving the entry at a later stage.

1. Issuing

Labels for existing records

For the records already in the database, labels with **link-based** QR codes can be printed via the Clipboard module. These codes, most importantly, contain the PlutoF internal ID, meaning that we can always map them to the specific record.

Reserved identifier block labels

The identifier blocks, available for creating in the Project Edit form, support a data gathering flow, in which the labels are generated before the data is entered, even before the material itself is gathered, which means that the QR code content cannot refer to a stable database ID.

These **entry** QR codes, then, are used to enter the new data. For example, a printed label is attached to a material bag, then scanned with the PlutoF GO app, filling the fields required to assign it to the allocated block (basis, identifier, project, collection).

Of course, the **entry** QR is not just for entering - after the data is in the DB, it works similarly to the **link-based** one, with the only difference being that it doesn't know the precise PlutoF record ID (since that did not exist at the moment of issuing). There is, however, enough information to resolve it.

2. Reading

Both types of codes can be scanned by GO to send the corresponding record to the Clipboard.

Entry codes can also be scanned in GO record specimen/material sample **add forms** to fill the identifier, project and collection. **Link-based** codes represent already existing data, so the validator will reject them here.

3. Code format

The collection identifier (ABC12345) does not uniquely point to the database record. At the very least, these are not unique between collections. For a proper link, an actual internal ID is required. In addition, the entry codes are part of a reserved block for either collection or a project, and that information also has to be included.

This means that just encoding the identifier does not suffice to support the intended flows. The QR code has to include additional fields.

These are encoded in a **PlutoF-specific** binary format, in addition to the format-describing metadata (for example, whether the code is a link or an entry). GO reads these, validates the format and gets all the required info from them (when used in a linking context, the internal ID still has to be resolved, of course).

4. Plaintext codes

This is all stable and the PlutoF flows work. However, using such a format means that the content is **not directly readable outside PlutoF**. Readers will get the binary bundle from the QR, but won't know what to do with it. The format is open, of course, so it can be decoded with some basic programming skills, but that is an extra step.

Depending on the use case, a **plain text** code may be preferable. These include **only the collection identifier**. This means that any QR reader, not just GO, will be able to get the identifier, and allows potentially using the codes outside the ecosystem (in some other application, or to simply get the identifier without having to type it).

Of course, the other side of using this is that **none of the PlutoF metadata is available**. In the linking context, we don't even know the basis of the record (is this a specimen or material sample?), which is why we have to ask the user (potentially mitigable in some cases - theoretically, we only need to ask if there is a collision, but I think the current implementation just prompts the user in any case).

Linking being just harder, **in the entry context**, a plaintext format loses most of the functionality of the binary one. If entering via a plaintext code, users would have to select the project/collection themselves. Though, if the post-entry lifetime usage outside PlutoF is deemed important enough, that is probably not a big deal.

At the moment, we **issue plain text codes** only for the user-configurable specimen labels (*"Specimen ID, with QR code"* format), but at some point the plan is to **allow users to select** between plaintext and rich (binary) codes before printing in other contexts.

Having plaintext codes as an option also means that since their introduction, GO now has to try and process any QR code fed into it. Before that, we could distinguish the PlutoF-generated ones by the format metadata. On a plus side, the code not generated by PlutoF (however those came to be), can also be read.

5. Addendum: the binary format specification

All 4-bytes integer fields are **little-endian**.
Basis is a one-byte enumeration.

Encoding	Basis
1	Specimen
2	Material sample
3	Living specimen

Entry format

Since these are all part of a reserved identifier block, the identifier format is very specific:

identifier = prefix + number padded to a certain format width

The prefix and the padding are defined for the project/collection the block comes from.

For example, a collection with prefix ABC and the format width 4 will have identifiers like ABC0123, ABC0023, ABC123456.

Field	Size in bytes	Content
Format magic number	2	ASCII “PF” (0x50, 0x46)
Format type	1	0x01 (entry format)
Basis	1	See the table above
Identifier prefix length	1	Length of the following prefix section
Identifier prefix	Variable	Identifier prefix, UTF-8 encoded
Format width	1	Length of the *padded* identifier number
Identifier number	4	Identifier number
Collection ID	4	
Project ID	4	

Link-based format

This is expected to work for any record, so the identifier is just included as a string (e.g “ABC0123”).

Field	**Size in bytes**	**Content**
Format magic number	2	ASCII “PF” (0x50, 0x46)
Format type	1	0x01 (entry format)
Basis	1	See the table above
Record ID	4	Internal PlutoF ID
Identifier length	1	Length of the following identifier section
Identifier	Variable	Identifier, UTF-8 encoded