Contents

% Better Together Community Engine Guide % Auto-generated (end_users) % \today

Overview

This PDF bundles the primary Community Engine documentation for end_users. Source of truth remains the repository markdown; diagrams should be re-rendered before export.

End User Guide

Welcome! This guide helps community members use Better Together platform features safely and effectively.

Getting Started

- User Guide - Your first steps on the platform
- Platform Welcome - Understanding Better Together principles

Community Features

- Exchange Process - Participating in exchanges and offers
- See also: Community Management for how communities operate

Safety and Privacy

- Privacy Principles - Protecting yourself and others
- Escalation Matrix - How serious issues are handled

Need Help?

Contact your community organizers or platform support staff for assistance with any platform features.

Better Together Community Engine Documentation Guide

This set of guides walks you through the core features of the Better Together Community Engine, organized under three main areas:

- Host Management
- Community Management
- Content Management

Use the links above to navigate to detailed instructions and reference materials for each feature.

 Welcome to Better Together!

Thank you for starting this journey with us. You're now setting up your very own Community Engine server—a powerful tool created with kindness, connection, and cooperation at its heart.

Better Together is more than technology. It's a movement dedicated to bringing people closer, so they can support each other and work together to solve important problems like climate change, fairness, and making communities stronger.

Our Community Engine helps you create a safe, welcoming space online. It's where your community can share ideas, plan projects, and take real action. It doesn't matter if you're new to tech or experienced—we believe everyone has something valuable to share.

Here's how we do it:

Discover new ideas, skills, and opportunities to help your community grow.

Connect deeply with others, building friendships and understanding.

Empower everyone to use their strengths, take action, and make positive changes.

We invite you to dream big. Imagine how much we can achieve when we listen to each other, support each other, and build together. This server is your first step towards creating a brighter, more connected future—where everyone feels valued, heard, and inspired to act.

Thank you for being part of this journey. We're excited to see the incredible things your community will accomplish!

Let's get started—because we're always Better Together.

Joatu Exchange: Process Overview

This document captures the current exchange system (Offers, Requests, Agreements, Matches, Response Links, Notifications) in a diagram-ready format.

Core Entities

- Offer: Something a person can provide.
  - Key fields: creator, name, description, categories, optional address, target (polymorphic), status (open/matched/fulfilled/closed), urgency (low/normal/high/critical), agreements, response links.
  - Behaviors: translated name/description, friendly slug, categorizable, view metrics, matches on create.
- Request: Something a person needs.
  - Same shape and behaviors as Offer.
- Agreement: Confirmation linking one Offer and one Request.
  - Status: pending/accepted/rejected.
  - Validations: Offer/Request target_type and target_id must align if present.
  - Side effects: on create → mark both sides matched if open; notify both creators. On accept → close Offer and Request; notify both creators. On reject → notify both creators.
- ResponseLink: Explicit link between a source (Offer or Request) and its opposite-type response (Request or Offer).
  - Constraints: opposite types only; source must be respondable (open or matched).
  - Side effects: mark source matched if it was open; symmetric notifications (Offer→Request notifies the offer creator; Request→Offer notifies the request creator); dedupe prevents duplicate unread notifications for the same Offer/Request pair.
- Matchmaker: Service that finds opposite-type matches.
  - Criteria: opposite type, status=open, category overlap (if any), same target_type; target_id rules: if one side has a specific id, the other side may have that id or nil (wildcard within the same target_type); exclude same creator; exclude pairs that already have a ResponseLink between the two; distinct results.

Actors & Permissions

- Creator: Creates/edits/closes own Offers/Requests; initiates Agreements; can accept/reject Agreements where they are a participant.
- Counterparty: Accepts/rejects Agreements they are party to.
- Platform manager: Elevated actions (policy-gated).
- Guests: Limited visibility; full exchange features require authentication.

State Machines

- Offer/Request status
  - open → matched: when a ResponseLink is created to/from it, or when an Agreement is created (callback tries to move open to matched).
  - matched → closed: when an Agreement is accepted (both sides closed).
  - open/matched → fulfilled: manual transition (separate from Agreement acceptance).
  - Any → closed: manual close possible (explicit enum state).
- Agreement status
  - pending: on creation.
  - pending → accepted: participant accepts; auto-closes Offer and Request.
  - pending → rejected: participant rejects; Offer/Request remain as-is.

Automatic Matching & Notifications

- When an Offer or Request is created, the system queries for matches using Matchmaker and sends a “New match found” notification to the creators of both sides for each match.
- Viewing an Offer/Request automatically marks its related match notifications as read for the current person. Viewing an Agreement marks its agreement-related notifications as read.

Primary Flows

1) Create Listing

- User creates an Offer or Request (valid name, description, 1 category; optional address; optional target polymorphic).
- Status=open.
- System runs Matchmaker; sends match notifications to involved creators; matches appear on listing pages (and Hub aggregates).

2) Direct Response (explicit link)

- From Offer Respond with Request: Prefilled Request form from Offer details; builds nested ResponseLink OfferRequest if source respondable; on create: mark Offer matched; notify offer creator via ResponseLink.
- From Request Respond with Offer: Prefilled Offer form from Request details; nested ResponseLink RequestOffer if source respondable (or controller fallback after save); on create: mark Request matched.

3) Agreement Lifecycle

- Initiation: Participant creates an Agreement linking a specific Offer and Request.
- Creation: Agreement status=pending; mark both sides matched (if open); notify both creators.
- Decision:
  - Accept: Agreement accepted Offer.status=closed; Request.status=closed; notify both creators.
  - Reject: Agreement rejected Offer/Request statuses unchanged (remain open or matched); notify both creators.

Permutations & Branch Points

- Direction: Offer-first or Request-first (symmetric).
- Target scope: target_type only; target_type + target_id; or both nil (general). Matching respects exactness: nil-only pairs with nil; id pairs require equality.
- Multiple Agreements: Multiple pending possible; acceptance of any one closes both sides, preventing further matches (Matchmaker filters on status=open).
- Response constraints: ResponseLink only if source is open or matched; otherwise blocked.
- Notifications: Match notifications on listing creation; ResponseLink one-way notify (OfferRequest); Agreement creation/status change notifications to both creators; auto mark-as-read on related page view.

Useful Routes (authenticated exchange scope)

- Exchange hub: GET /:locale/bt/exchange
- Offers: CRUD; GET /exchange/offers/:id/respond_with_request
- Requests: CRUD; GET /exchange/requests/:id/respond_with_offer; GET /exchange/requests/:id/matches
- Agreements: CRUD; POST /exchange/agreements/:id/accept; POST /exchange/agreements/:id/reject

Diagram Notes

- Use swimlanes for Offer Creator, Request Creator, System (optional in Mermaid; can represent via subgraphs).
- Group flows: Create Listing, Direct Response, Agreement Lifecycle, Notification Read, and State Transitions.
- Show key decision nodes: respondable source? agreement accepted? target alignment validation.

Event Invitations and RSVP (End Users)

This guide explains how to view and respond to event invitations, how RSVP works, and what to expect on private platforms.

Receiving Invitations

- You may be invited to an event either as a member (in-app notification + optional email) or via email (external address).
- Invitation links include a secure token that lets you review the event details and respond.
- If you aren't signed in, you'll be redirected to sign in or register; your invitation is saved so you can continue after authentication.

Reviewing Invitations

- Invitation review page: shows event name, description, and your invitation status.
- Private platforms: your invitation link grants access to the specific event page even if the rest of the platform is private.
- Language: the invitation is localized to the language selected by the inviter; your session will switch to that language on first visit.

Responding to Invitations

- Accept: you'll be added as "Going". If needed, you'll be added to the host community automatically.
- Decline: marks the invitation declined; you can still view public details of the event if available.
- After accepting, your RSVP will appear as "Going" and a calendar entry will be added to your personal calendar.

RSVP Controls on Event Pages

- Interested: marks your interest without committing to attend.
- Going: confirms attendance; creates a calendar entry.
- Cancel: removes your attendance record and deletes the related calendar entry.
- RSVP is available only after the event is scheduled (has a start time).

ICS Calendar Export

- Event pages offer a ".ics" export (Add to Calendar) once scheduled.
- If you RSVP "Going", your personal calendar also gets a calendar entry automatically.

Privacy and Access

- Invitation tokens are limited to the specific event; they don't grant access beyond it.
- If your token is expired or invalid, you'll be asked to sign in or may see a not found page.

Troubleshooting

- Link not working: check if the invitation has expired or was already accepted/declined.
- Can't RSVP: the event might still be a draft (no start time). Try again once it's scheduled.
- No calendar entry: only "Going" creates an entry; "Interested" does not. Make sure you have a primary calendar set up.
- Wrong account: if the invitation is for a specific member account, log in with that account to accept.