ChatGPT

Protocol for Mutual Document Linking in QIKI_DTMP

Purpose

This protocol defines how every technical and task document in the **QIKI_DTMP** project must cross-reference other files. The goal is to **preserve context**, **avoid duplication**, and **enable agents to recover state quickly** after a pause or context switch. By embedding explicit forward and backward links, changes in one place can be traced throughout the system. This document complements the Task Execution System and expands upon the mandatory cross-link fields mentioned by the user.

Guiding Principles

1. **Bidirectional Linking** – Each document must know who depends on it and whom it depends upon. This follows the anti-duplication rules in the task system.
2. **Honest Documentation** – Links must reflect reality. Do not list tasks or documents unless they actually use or depend on this information. This aligns with the principle of absolute transparency.
3. **Minimalism and Clarity** – Keep the linking sections concise. Long descriptions belong in the main body; cross-links are for navigation and dependency tracking.
4. **Automatic Verification** – Whenever possible, scripts or checklists should verify that referenced documents exist and that reverse links are reciprocated. This prevents broken references and lost context.

Required Sections for Technical Documents

Every technical document (design docs, architecture specs, protocol definitions, etc.) must include the following fields at the end of the document:

Связанные задачи (Related Tasks)

List all **Task** documents that directly use or implement the information from this technical document. Include the task filename and a brief phrase describing the connection. This helps future maintainers see where the design has been applied.

Format:

```
## Связанные задачи
- [TASK_YYYYMMDD_FIX_IMPORTS.md] – uses section “Структура компонентов” to correct imports
- [TASK_YYYYMMDD_SETUP_TESTING.md] – implements testing strategy defined here
```

Зависимые документы (Dependent Documents)

List documents that **must be updated** when this document changes. These might include other design documents, generated schemas, or master indexes. When editing this file, you must open each dependent document and update relevant sections accordingly. This ensures that changes propagate across the documentation tree and prevents inconsistencies.

Format:

```
## Зависимые документы
- [FILEDEVELOPER.md] – section “2. Structure” references this specification
- [IMPLEMENTATION_ROADMAP.md] – update the roadmap if architectural changes
occur
```

↩ Обратные ссылки (Back-references)

Describe where else in the documentation this information is referenced. These are not tasks that implement the design, but rather other documents that mention this concept. Maintaining back-references helps prevent duplication and makes it easier to track the spread of ideas across the project. When you add a back-reference here, ensure that the other document’s **Зависимые документы** or **Связанные задачи** sections include a reciprocal link.

Format:

```
## Обратные ссылки
- [CLAUDE_PROJECT_ANALYSIS.md] – subsection 6.3 “Protocol Buffers Contracts”
cites this design
- [AI_DEVELOPER_PRINCIPLES.md] – principle “Contract-Oriented Architecture”
inspired by this spec
```

Required Sections for Task Documents

Every **Task** document must include a header that links back to its sources and obligations. This is critical for understanding the scope of a task and ensuring that required updates occur upon completion.

📚 Базовые документы (Base Documents)

Enumerate all technical documents that provide the requirements or constraints for this task. If you are implementing a feature from a design doc, specify the relevant sections. If you are fixing a bug, link to the specification that defines the intended behaviour.

Format:

```
## БАЗОВЫЕ ДОКУМЕНТЫ
- [ACTUATOR_RAW_OUT.proto] – defines command structure; see “retry_count” and
```

```
"timeout_ms"
- [IMPLEMENTATION_ROADMAP.md] – Phase 1, task T1.1
```

Конкретные разделы (Specific Sections)

Pinpoint the exact paragraphs, sections, or function names you will be working on. Avoid vague references; cite section numbers, headers, or code identifiers. This precision prevents misinterpretation and helps the reviewing agent navigate quickly to the relevant context.

Format:

```
## КОНКРЕТНЫЕ РАЗДЕЛЫ
- "3.3 Критические Блокеры" in IMPLEMENTATION_ROADMAP.md – describes the
import bug to fix
- `fsm_handler.py::FSMStateEnum` – update enum names according to spec
```

Обязанности по обновлению (Update Obligations)

When the task completes, state which documents must be updated and what fields must be modified. For example, if you implement a design specification, you may need to tick off criteria in the roadmap, adjust readiness percentages in **CLAUDE_MEMORY.md**, or add a new lesson to **LESSONS_LEARNED.md**. By listing these obligations upfront, you ensure that context preservation rules are followed.

Format:

```
## ОБЯЗАННОСТИ ПО ОБНОВЛЕНИЮ
- Update CURRENT_STATE.md – change status of "Generated Code Imports" to
"Fixed"
- Append DECISIONS_LOG.md – record the chosen fix method
- Update CLAUDE_MEMORY.md – increment project readiness from 47% to 55%
```

Workflow Recommendations

1. **Pre-Task Checklist** – Before creating a new task, execute the pre-task checklist defined in the Task Execution System (search existing tasks, decisions, lessons). This helps detect duplicates and ensures you link to the correct base documents.
2. **Context Recovery** – When resuming work after a pause, read **CLAUDE_MEMORY.md**, the last three task files, and **CURRENT_STATE.md**. Use the back-references and dependent document lists to quickly rebuild context.
3. **Reciprocal Linking** – Whenever you add a reference in one document, update the reciprocal section in the referenced document. For example, if a task lists `IMPLEMENTATION_ROADMAP.md` as a base document, add this task to the roadmap's **Связанные задачи** section. Scripts can automate this consistency check.
4. **Version Control** – Treat document updates as code changes: commit with clear messages, review, and test for broken links. Consider writing a simple linter to verify that each document's dependent files exist and that referenced sections are valid.

5. **Lessons Learned Integration** – At the end of each task, summarise any pitfalls or insights in **LESSONS_LEARNED.md** and link back to the task file. This builds a knowledge base to avoid repeating mistakes and aligns with the anti-duplication patterns described in the Task Execution System.

Future Considerations

Automated tooling – The linking system can be enhanced with a small script that scans Markdown headers and file names, checks that mutual references are present, and generates reports of missing links. Integrating this script into your CI pipeline ensures that documentation stays consistent as the project evolves.

Scalability – As the number of documents grows, maintain an **INDEX.md** or use your documentation generator (qiki_docgen) to create an overview page listing all documents with their linked tasks and dependencies. This will help agents navigate large sets of files efficiently.

Conclusion

By rigorously applying the cross-linking rules described here, you will create a **self-navigating documentation ecosystem**. Agents waking from sleep will always know where to go next; changes in one place will ripple through dependent documents; and the history of decisions, lessons, and tasks will remain coherent. Combined with the honesty and incremental practices laid out in **AI_DEVELOPER_PRINCIPLES.md** and the structured workflow in **TASK_EXECUTION_SYSTEM.md**, this protocol forms a solid foundation for the long-term success of QIKI_DTMP.
