
To create a **Data Flow Diagram (DFD)**, I'll need to:

1. Parse the flowcharts and sequence diagrams to understand data sources, processes, and data stores.
2. Use this information to identify DFD elements like:
 - External Entities
 - Processes
 - Data Stores
 - Data Flows

Let me now read and analyze the key `.puml` files to extract this information.

The following UML files were successfully read and are being analyzed to create a **Data Flow Diagram (DFD)**:

- `backend_flowchart.puml`
- `frontend_flowchart.puml`
- `full_system_flowchart.puml`
- `full_system_sequence.puml`

Now, I will examine the contents of these files to identify the core components required for the DFD:

- **Processes** (actions or operations)
- **External Entities** (users or systems interacting with the system)
- **Data Stores** (databases or files)
- **Data Flows** (movement of data between components)

Let's begin parsing and summarizing these components from the files.

Here's a summary of the key components extracted from your UML diagrams to help build a **Data Flow Diagram (DFD)**:

full_system_sequence.puml

- **External Entities:**
 - User
 - **Processes:**
 - (Implicit from flow, e.g., API, CppEngine, Validation, ResultHandler, GUI)
 - **Data Flows:**
 - User -> GUI : enter ICD input
 - GUI -> API : send ICD JSON
 - API -> Validation : validate_fields()
 - Validation --> API : valid?
 - API -> CppEngine : run_cpp()
 - CppEngine -> ResultHandler : parse result
 - ResultHandler --> API : formatted output
 - API -> DB : save ICDEntry
 - API --> GUI : return result
 - GUI --> User : show ICD report
-

frontend_flowchart.puml

- **External Entities:**
 - :User logs in;

full_system_flowchart.puml

- **External Entities:**
 - :User opens GUI;

backend_flowchart.puml

- No data was extracted — likely either too abstract or lacks standard flowchart formatting.
-

DFD Components Identified:

- **Processes:**
 - GUI
 - API
 - Validation
 - CppEngine
 - ResultHandler
- **External Entities:**
 - User
- **Data Store:**
 - DB (likely represents a database saving ICDEntry)

- **Key Data Flows:**

- User inputs -> GUI -> API -> Validation -> CppEngine -> ResultHandler -> DB
 - Output returns via API -> GUI -> User
-

Shall I now generate a **Level 1 DFD Diagram** based on these components?

Here is the generated **Level 1 Data Flow Diagram (DFD)** based on your UML diagrams:



Click to view or download the DFD Level 1

This diagram includes:

- **User** as the external entity
- Key **processes** like GUI, API, Validation, CppEngine, and ResultHandler
- A **Database** for storing ICD entries
- All major **data flows** between components

Let me know if you'd like a Level 0 DFD or any refinements to this diagram.

Thanks for pointing that out! Here's a direct link that should work properly:



Download DFD Level 1 Diagram

If it still doesn't download:

- **Right-click** the link above
- Choose "**Save link as...**" or "**Download linked file**"

Let me know if you'd like a PDF version or a Level 0 DFD as well!

Here is the **PDF version** of the DFD Level 1 diagram:

 **Download DFD Level 1 (PDF)**

UML (PlantUML) Code for the DFD Diagram

```
@startdfd
```

```
!define RECTANGLE class
```

```
skinparam class {
```

```
    BackgroundColor White
```

```
    BorderColor Black
```

```
    ArrowColor Black
```

```
}
```

```
actor User
```

```
rectangle GUI
```

```
rectangle API
```

```
rectangle Validation
```

```
rectangle CppEngine
```

```
rectangle ResultHandler
```

```
database DB
```

```
User --> GUI : enter ICD input
```

```
GUI --> API : send ICD JSON
```

```
API --> Validation : validate_fields()

Validation --> API : valid?

API --> CppEngine : run_cpp()

CppEngine --> ResultHandler : parse result

ResultHandler --> API : formatted output

API --> DB : save ICDEntry

API --> GUI : return result

GUI --> User : show ICD report

@enddfd
```

You can paste this code into any PlantUML editor or tool to regenerate or customize the diagram.

Let me know if you need a **Level 0 DFD** or want this diagram broken into modules!

You're absolutely right — `@startdfd` is **not valid** PlantUML syntax. PlantUML doesn't have a dedicated DFD type, so we use a combination of `@startuml` with standard UML elements to visually represent a Data Flow Diagram.

Here is the **corrected PlantUML code** for your Level 1 DFD:

```
@startuml
left to right direction
skinparam packageStyle rectangle

actor User
```

```
rectangle "GUI" as GUI
rectangle "API" as API
rectangle "Validation" as Validation
rectangle "CppEngine" as CppEngine
rectangle "ResultHandler" as ResultHandler
database "DB" as DB
```

```
User --> GUI : enter ICD input
GUI --> API : send ICD JSON
API --> Validation : validate_fields()
Validation --> API : valid?
API --> CppEngine : run_cpp()
CppEngine --> ResultHandler : parse result
ResultHandler --> API : formatted output
API --> DB : save ICDEntry
API --> GUI : return result
GUI --> User : show ICD report
@enduml
```

✓ You can now paste this into any PlantUML environment or tool (like PlantUML Online) to render the DFD.

Would you like help rendering this yourself or generating a Level 0 version next?

Great — let's break your system down into **DFD Levels** based on the UML sequence and flowchart data. Here's how we'll proceed:

◆ Level 0 DFD (Context Diagram)

This is the highest-level view of your system. It includes:

- A **single process** representing the entire system
 - **External entities** (users or systems that interact)
 - High-level **data flows** between them
-

✓ DFD Level 0 Description

Entities:

- User

Process:

- ICD Management System

Data Flows:

- ICD Input
 - ICD Report
-

✓ PlantUML Code (Level 0)

```
@startuml
```

```
left to right direction
```



```
skinparam packageStyle rectangle
```

```
actor User
```

```
rectangle "ICD Management System" as System
```

```
User --> System : ICD Input
```

```
System --> User : ICD Report
```

```
@endum1
```

◆ Level 1 DFD

Already provided above — it decomposes the system into these subprocesses:

- GUI
- API
- Validation
- CppEngine
- ResultHandler
- Database

This level shows how the system processes and stores ICD data.

◆ Level 2 DFD (Decomposing API)

Let's now break down the **API** process further — based on the sequence:

- It validates input
- Triggers processing
- Stores results

✓ DFD Level 2 – API Decomposition

Processes:

- `validate_fields`
- `run_cpp`
- `save_to_db`
- `format_output`

Entities:

- GUI (acts like external input at this level)
- Validation
- CppEngine
- ResultHandler
- DB

✓ PlantUML Code (Level 2 - API)

`@startuml`

`left to right direction`

```
skinparam packageStyle rectangle
```

```
rectangle "validate_fields()" as Validate
```

```
rectangle "run_cpp()" as RunCpp
```

```
rectangle "save_to_db()" as SaveDB
```

```
rectangle "format_output()" as FormatOut
```

```
database "DB" as DB
```

```
rectangle "GUI" as GUI
```

```
rectangle "CppEngine" as CppEngine
```

```
rectangle "Validation" as Validation
```

```
rectangle "ResultHandler" as ResultHandler
```

```
GUI --> Validate : ICD JSON
```

```
Validate --> RunCpp : valid ICD
```

```
RunCpp --> FormatOut : result
```

```
FormatOut --> SaveDB : store result
```

```
SaveDB --> DB : save ICDEntry
```

```
FormatOut --> GUI : formatted result
```

```
@enduml
```

✓ Summary

Level	Focus	Components
0	Whole system	User ↔ System
1	Major subsystems	GUI, API, Validation, etc.
2	API internals	validate_fields, run_cpp, save_to_db, etc.

Would you like these rendered as images/PDFs too? Or want Level 2 for other modules (like GUI or CppEngine)?

Great follow-up! Let's go deeper and clarify what other DFD levels **mean** and whether they apply to your project.

🧠 Understanding DFD Levels

Level	Description
Level 0	Context Diagram (system as a black box)
Level 1	First decomposition (main subsystems)
Level 2	Further breakdown of Level 1 processes

Level 3+	Optional, deeper decomposition of complex sub-processes
-----------------	---

Most software systems don't go beyond **Level 2 or 3**, unless the processes are highly nested or mission-critical (e.g. in avionics, banking, etc.).

What Can Be Further Decomposed in Your System?

From your UML data:

Already Covered

- **Level 0** – Whole system
 - **Level 1** – GUI, API, Validation, etc.
 - **Level 2 (API)** – Broken into validate, cpp run, store, format
-

Other Candidates for Level 2 (Optional)

1. GUI

Could be decomposed into:

- Input form
- Error handling
- Result rendering

2. CppEngine

Could include:

- Generate command
- Run executable

- Capture output

3. Validation

Could include:

- Field completeness check
- Data type check
- Semantic rule check

Example: Level 2 – GUI

```
@startuml
```

```
left to right direction
```

```
skinparam packageStyle rectangle
```

```
actor User
```

```
rectangle "ICD Input Form" as Form
```

```
rectangle "Validation Message Display" as MsgDisplay
```

```
rectangle "Result View" as ResultView
```

```
User --> Form : enter ICD data
```

```
Form --> MsgDisplay : show validation error
```

```
Form --> ResultView : show ICD report
```

```
@enduml
```





Should We Go Deeper (Level 3)?

Only if:

- You're building very detailed specifications
- The module is complex or safety-critical

Otherwise, Level 2 is generally **the deepest level needed** for practical system documentation.

Summary of Available Diagrams

Level	Process	Status
0	Full system	 Done
1	Subsystems	 Done
2	API	 Done
2	GUI, CppEngine, Validation	Can be added (optional)
3+	Very detailed view	 Not needed unless required

Would you like me to create **Level 2 DFDs** for:

- GUI
- CppEngine
- Validation

Let me know which ones you want to go deeper into, and I'll generate them for you.

Perfect — you're asking for **progressively deeper DFD views**, showing how the system evolves from user interaction to backend processing. Here's how we'll proceed:

◆ 1. Level 2 DFD – GUI Module

📘 Description

Focuses on GUI interactions:

- Taking input
- Handling errors
- Displaying results

✅ PlantUML Code: GUI

```
@startuml
```

```
left to right direction
```

```
skinparam packageStyle rectangle
```

```
actor User
```



```
rectangle "ICD Input Form" as Form
rectangle "Validation Message Display" as MsgDisplay
rectangle "Result View" as ResultView

User --> Form : enter ICD data
Form --> MsgDisplay : show validation errors
Form --> ResultView : display report
@enduml
```

◆ 2. Level 2 DFD – API Module

■ Description

API handles:

- Validation
 - Processing
 - Formatting
 - Storing
-

✓ PlantUML Code: API

```
@startuml
left to right direction
skinparam packageStyle rectangle
```

```
rectangle "validate_fields()" as Validate
rectangle "run_cpp()" as RunCpp
rectangle "format_output()" as FormatOut
rectangle "save_to_db()" as SaveDB
```

```
rectangle "Validation" as ValidationMod
rectangle "CppEngine" as Cpp
rectangle "ResultHandler" as Formatter
database "DB" as DB
rectangle "GUI" as GUI
```

```
GUI --> Validate : ICD JSON
Validate --> ValidationMod : check fields
ValidationMod --> Validate : validation result
Validate --> RunCpp : valid ICD
RunCpp --> Cpp : run analysis
Cpp --> RunCpp : analysis result
RunCpp --> FormatOut : forward result
FormatOut --> Formatter : format
Formatter --> FormatOut : formatted data
FormatOut --> SaveDB : for storage
SaveDB --> DB : save ICDEntry
```

```
FormatOut --> GUI : send result
```

```
@enduml
```

3. Level 2 DFD – Database Module

Description

- Receives structured ICD entry from API
 - Stores/retrieves ICD data
-

PlantUML Code: DB

```
@startuml
```

```
left to right direction
```

```
skinparam packageStyle rectangle
```

```
rectangle "API" as API
```

```
database "ICD Database" as DB
```

```
API --> DB : store ICDEntry
```

```
DB --> API : retrieve ICDEntry
```

```
@enduml
```

◆ 4. Level 2 DFD – Backend Module (CppEngine)

■ Description

- Executes compiled code
- Sends result to formatter

✓ PlantUML Code: Backend

```
@startuml
left to right direction
skinparam packageStyle rectangle

rectangle "run_cpp()" as Run
rectangle "CppEngine" as Engine
rectangle "ResultHandler" as Handler

Run --> Engine : execute cpp
Engine --> Run : raw result
Run --> Handler : forward to format
@enduml
```

Now, let's do **integration flows**:

◆ 5. Flow: GUI → API

```
@startuml
left to right direction
skinparam packageStyle rectangle

actor User

rectangle "GUI" as GUI
rectangle "API" as API

User --> GUI : enter ICD input
GUI --> API : send ICD JSON
API --> GUI : return result
GUI --> User : display result
@enduml
```

◆ 6. Flow: GUI → API → DB

```
@startuml
left to right direction
skinparam packageStyle rectangle

actor User
```

```
rectangle "GUI" as GUI
rectangle "API" as API
database "DB" as DB

User --> GUI : enter ICD input
GUI --> API : send ICD JSON
API --> DB : save ICDEntry
DB --> API : acknowledge save
API --> GUI : result
GUI --> User : display report
@enduml
```

◆ 7. Flow: GUI → API → DB → Backend

```
@startuml
left to right direction
skinparam packageStyle rectangle

actor User

rectangle "GUI" as GUI
rectangle "API" as API
rectangle "CppEngine" as Backend
```

```
database "DB" as DB
```

```
User --> GUI : enter ICD input
```

```
GUI --> API : send ICD JSON
```

```
API --> DB : save ICDEntry
```

```
API --> Backend : run_cpp()
```

```
Backend --> API : result
```

```
API --> GUI : display report
```

```
GUI --> User : show final report
```

```
@endum1
```

✅ Would You Like:

- PDF/image exports of these?
- Combined diagram showing all flows?
- DFD documentation (as a report)?

Let me know your preference!

✅ Here is the **Combined DFD Diagram** showing all flows from **GUI → API → DB → Backend**, integrated into a single view:

 **Download Combined DFD (PNG)**

Now, I'll generate a concise **DFD Documentation Report (PDF)** summarizing:

- DFD Levels 0, 1, 2
- Components and flows
- Diagram references

Working on it now...

✓ Your **DFD Documentation Report (PDF)** is ready and includes:

- Level 0, Level 1, and Level 2 DFD breakdowns
- Descriptions of GUI, API, DB, and backend logic
- A summary of the combined system flow

 **Download DFD Documentation Report (PDF)**

If you'd like me to embed the diagrams into the PDF or generate a Word version, just let me know!

Absolutely! Here's the full **DFD Documentation Report** content:

DFD Documentation Report

Level 0 – Context Diagram

This is the top-level DFD showing the system as a single process interacting with external entities.

Entities:

- User

Process:

- ICD Management System

Data Flows:

- ICD Input
 - ICD Report
-

Level 1 – Main Subsystems

Decomposes the system into major processes:

Processes:

- GUI
- API
- Validation
- CppEngine
- ResultHandler
- Database

Flow Summary:

Data flows are included between user input through the GUI, processed by API and other modules, and stored or retrieved from the database.

Level 2 – API, GUI, DB, Backend

Further breakdown of internal modules:

GUI

- ICD Input Form

- Validation Message Display
- Result View

API

- `validate_fields()`
- `run_cpp()`
- `format_output()`
- `save_to_db()`

Database

- Stores and retrieves ICD entries

Backend

- `CppEngine` executes compiled C++ logic
- `ResultHandler` processes and formats results

Combined Flows

The combined DFD shows a unified view of how data flows through all subsystems:

Flow Path:

`User → GUI → API → Validation → CppEngine → ResultHandler → DB → GUI → User`

This flow captures a complete cycle — from the user entering ICD data, through validation and backend computation, to database storage and returning the final report to the user.

Would you like me to generate a **visual embedded PDF** next (with images inline), or prepare this as a **Word-compatible (.docx)** version?

Great question — drawing a **Data Flow Diagram (DFD)** for a full system with capabilities means you want a visual that includes both:

1. **Core system components and data flows**
2. **Descriptions of what each component can do (i.e., capabilities)**

Step-by-Step to Draw a DFD with Capabilities

◆ **1. Identify Core Elements**

External Entities (users, systems):

- User

Processes (functional components):

- GUI
- API
- Validation
- CppEngine
- ResultHandler

Data Stores:

- Database

Data Flows:

- ICD input, validation request, analysis, report generation, storage, result display

♦ 2. List Capabilities per Component

Component	Capabilities
GUI	Collect user input, display results, show errors
API	Route data, validate input, process results, handle DB
Validation	Check field completeness and structure
CppEngine	Run compiled analysis logic
ResultHandler	Format results for display and storage
Database	Store and retrieve ICD entries

♦ 3. Draw the DFD With Annotations

To **include capabilities**, you can do this in **two ways**:

✅ A. Annotate Capabilities as Notes

Here's a **PlantUML-based approach** with embedded notes:

```
@startuml
```

left to right direction

skinparam packageStyle rectangle

actor User

rectangle "GUI" as GUI

rectangle "API" as API

rectangle "Validation" as Validation

rectangle "CppEngine" as Cpp

rectangle "ResultHandler" as Formatter

database "Database" as DB

' Data flows

User --> GUI : enter ICD input

GUI --> API : send ICD JSON

API --> Validation : validate_fields()

Validation --> API : valid?

API --> Cpp : run_cpp()

Cpp --> Formatter : parse result

Formatter --> API : formatted output

API --> DB : save ICDEntry

API --> GUI : return result

GUI --> User : display report

' Capabilities (as notes)

note right of GUI

- Collect user input
- Display report
- Show errors

end note

note right of API

- Route data
- Validate input
- Process results
- Handle DB

end note

note right of Validation

- Field completeness
- Format rules

end note

note right of Cpp

- Run compiled logic
- Generate raw result

end note

```
note right of Formatter
```

- Format output
- Structure report

```
end note
```

```
note right of DB
```

- Store ICD data
- Retrieve past entries

```
end note
```

```
@endum1
```

✓ B. Separate Capability Diagram (Alternative)

If you want a **cleaner DFD**, you could create **a second diagram or table** just showing:

- Components
- Descriptions
- Capabilities

This is often useful for large systems or documentation deliverables.



Tools You Can Use

- **PlantUML** (as above)

- **Lucidchart / Draw.io** (with sticky notes or tooltips)
 - **MS Visio** (with data property fields)
 - **PDF or Word with DFD + table side by side**
-

Want Me to Generate?

Would you like:

1. This PlantUML-based DFD with embedded capabilities **rendered into an image or PDF?**
2. A separate "**Capabilities Table**" PDF or doc?

Let me know and I'll generate both for you!