Upgrade and Designer Activities

Upgrade Activities: Any Version ➔ E1U3 ➔ E2U1/2

Link: Upgrade Steps & Sequence to follow for transforming M7 to E2 - Team Platform - Confluence

Form Transformation

- If forms are extended, they must undergo transformation using **TUC (Transformation Utility Console)**.
- During the transformation process, you can choose between a tab or section as the layout type.
- Convert:
    - **JavaScript (JS)** to **Business Rules (BR)** → BR User Guide
    - **Backbone JS** to **React** using **mPower API**.
    - **PLSQL** to **Emery (Groovy equivalent)** → This is not a mandatory step, but if any new changes are implemented, suggestions to use Groovy only.
- Merge business rules if applicable, for extended forms to incorporate the (OOTB) form rules.
- Reusable components/widgets (e.g., password fields, calculators, sections) should be used if available. If not, request creation from the **Platform UX Team**.
- **BAPI/SDU** support should be ensured.
- **Workflow stage indicator** conditions should be moved from JavaScript (JS) to Business Rules (BR) or the Infolet level. → https://metricstream.atlassian.net/wiki/spaces/TP/pages/53783900/Configuring+Workflow+Stage+Indicator+in+Forms).

  Additional Field Impacts: Forms, Reports, Charts/Dashboards, BAPI/SDU.

Report Transformation

- Reports not **upgrade-safe** require:
    - Backbone JS to React migration using **R-API**.
    - Merging changes from Out-of-the-Product (OOTP) reports with customized ones, or creating new reports by merging both.
    - Additional JS transformations.
    - BI-View Changes

Chart & Dashboard Transformation

- Charts are not upgrade-safe require:
    - Conversion and migration of Backbone JS to React using R-API.

- Rebuilding using REST APIs (Postman payloads) if needed.
- Reusable components wherever possible, else involve the **Platform UX Team**.

- Support customizations or re-creations based on the merged logic of OOTP and customized versions.

Infocenter Updates

- Layout changes are required.
- Apply customizations by porting changes or creating new versions as necessary.

Other Areas to Review

- **SSR (Self-Service Reporting):** Validation is needed if transformation is required.
- **Calendar:** Check and adjust customizations, if any.

Change Scenarios

New Field Additions

- **Impact Areas**: Data Object (DO), Forms, Reports, Charts/Dashboards, Emails, Workflows, BAPI/SDU (other than hidden fields), Infolet/AJAX calls, Business Rules, Emery (Groovy), Help pages, ORF changes (if org field involved/impacted), BI Views.

Label Changes

- Form updates (supported by Dev Utility tools).

Hiding Field Changes

Other Changes

Product Development & Enhancements

Typical Use of Platform Designers

- Create **Forms (**including dependent items**), Reports, Workflows, Charts, and Scorecards** based on business use cases and functional requirements.

Change Scenarios

New Field Additions

- **Impact Areas**: Data Object (DO), Forms, Reports, Charts/Dashboards, Emails, Workflows, BAPI/SDU (other than hidden fields), Infolet/AJAX calls, Business Rules, Emery (Groovy), Help pages, ORF changes (if org field involved/impacted).

Label Changes

- Form updates (supported by Dev Utility tools).

If Forms/Components are Not Transformed

- They must be transformed using **TUC (Transformation Utility Console)**.
- Convert:
    - **JavaScript (JS)** to **Business Rules (BR)**.
    - **Backbone JS** to **React** using **mPower API**.
    - **PLSQL** to **Emery (Groovy equivalent)**.
- Merge business rules if applicable.
- Reusable components/widgets (e.g., password fields, calculators, sections) should be used if available. If not, request creation from the **Platform UX Team**.
- **BAPI/SDU** support should be ensured.
- Support **Workflow (WF) Stage Indicator** enhancements.

Platform Expectation During Any Upgrade

- Products **consume platform builds** in **Dev** and **Staging** (including MI upgrades or service reinstalls).
- Backward compatibility is **mandatory** unless there's a **platform design change** communicated.
- If the platform mandates product-side actions (e.g., re-exporting chart IUPs), **products must perform those activities**.

Upgrade to E2 Specific: Product Activities

- Consume Platform builds as above.
- Perform:
    - Form Transformation
    - Report Transformation
    - Chart and Scorecard Development
    - Infocenter Layout Changes

□ Observations to Find Common Patterns (for next step)

Common tasks across both Upgrade and Product Development:

- Form, Report, and Chart/Dashboard transformation or development.
- JS to BR and Backbone to React migrations.
- PLSQL to Emery (Groovy) conversion.

- Usage of reusable components or widget requests to the UX team.
- BAPI/SDU handling.
- Workflow and Infolet updates.
- Support for platform-driven changes and backward compatibility.

Key Pain Points Identified

Category	Area	Description	Impact / Ask

Category	Area	Description	Impact / Ask
BAPI/SDU Complexity	**BAPI Developer Journey**	• High effort is required for every field change in BAPI/SDU. • The current process is time-consuming and repetitive.	**One-Click BAPI Dev Journey:** Simplify BAPI changes with automation and reduce manual effort.
Chart Complexity	**Chart**	• Modifying charts is complex and not user-friendly. • Requires multiple manual steps.	**Simplification Needed:** Create a user-friendly interface or tool to easily modify charts and dashboards.
Debugging Challenges	**Groovy Scripts**	• No dedicated tool for writing or debugging Groovy scripts. • Debugging is runtime-based (no compiler). • Hard to trace errors or line numbers in client logs without adding multiple print statements.	• **Need a Groovy Debugger or Utility** to ease debugging. • Should support breakpoints and line tracing to reduce developer effort.
Logging Improvements	**Logs (Client & DB)**	• Logs are cluttered with unwanted errors. • Tracing the root cause requires sifting through multiple logs (client, DB). • Troubleshooting may take more than a day.	• **Consolidated Logging:** A single log placeholder combining DB & Client logs. • Simplify debugging by filtering relevant logs.
Form Submission Experience	**Forms**	• Failed form submissions leave users unaware of the issue. • End users rely heavily on MSI/Support to investigate and resubmit.	• **Graceful error handling:** Show meaningful error messages. • Allow users to correct and resubmit themselves.
Integration Usability	**CSV/Excel Uploads**	• Multiple inconsistent programs exist across projects for handling CSV/Excel uploads. • Lack of a unified approach or tool for data ingestion.	• **Common Upload Framework:** UI-based mapping tool to upload CSV/Excel to staging tables. • Project teams define the logic post-upload.