

Therefore™ eForms – Design Principles

Product Positioning, Customisation and Solution Architecture Guidance

Therefore™ eForms provides a flexible and powerful way to capture structured information and connect users with document-centric business processes in Therefore.

eForms supports a wide range of scenarios, from straightforward electronic forms to more sophisticated forms incorporating validation, conditional behaviour and customer-specific logic. This flexibility is an important part of the value of eForms and enables partners and customers to tailor solutions to individual business requirements.

As solutions become more sophisticated, however, it is important to distinguish between **customising an electronic form** and **developing a business application**.

This guidance clarifies the intended role of Therefore eForms, provides guidance on the appropriate use of customisation and scripting, and describes the recommended architecture when requirements extend beyond form-based data capture.

The Intended Role of Therefore™ eForms

Therefore™ eForms is designed primarily for **form-based data capture and form-driven interaction with Therefore processes**.

Typical scenarios include:

- Replacing paper or static forms with digital forms;
- Capturing structured information from employees, customers, suppliers or other users;
- Validating information before it is submitted;
- Providing conditional form behavior based on information entered by the user;
- Collecting information and supporting documents;
- Initiating or participating in processes managed within Therefore™;
- providing lightweight form-based user interfaces for specific interactions with Therefore™;
- Supporting anonymous or embedded data-entry scenarios where appropriately configured.

Within these scenarios, eForms can provide a rich and highly configurable user experience.

eForms also provides customization capabilities that allow partners and customers to extend standard form behavior where appropriate.

However, **Therefore™ eForms is not intended to be a general-purpose low-code application development platform or a replacement for a dedicated application development framework.**

This distinction becomes increasingly important as a solution grows in complexity.

Form customisation versus application development

Many business requirements can be addressed very effectively using an eForm. A typical eForms scenario follows a relatively straightforward pattern:

Capture information → validate → submit → continue the business process in Therefore™

Examples include:

- Service requests;
- Application and registration forms;
- Employee onboarding information;
- Inspections and checklists;
- Questionnaires;
- Information and document submission;
- Process initiation;
- Structured data-entry interfaces.

Some customization and scripting can be entirely appropriate within these scenarios. The architecture should be reconsidered, however, when the primary purpose of the solution starts to change from **capturing information** to **operating a business application**.

Typical indicators include:

- Extensive JavaScript or other client-side logic;
- Sophisticated user-interface behaviour;
- Complex application-style navigation;
- Significant amounts of business logic implemented within the form;
- Substantial application state;
- Extensive interaction between custom components;
- Significant dynamic manipulation of the user interface;
- Multiple external integrations orchestrated through client-side scripts;
- Critical business functionality that depends heavily on custom form scripts or implementation-specific behaviour.

These characteristics do not necessarily mean that a solution cannot technically be implemented using eForms.

They are, however, indicators that a dedicated application architecture may provide a more maintainable and sustainable solution.

A useful design principle is:

If the primary purpose is collecting information, eForms is likely to be the appropriate user interface. If the primary purpose is running a business application, a dedicated application architecture should be considered.

Customisation and JavaScript

Therefore™ eForms provides scripting and customization capabilities that enable forms to be adapted to specific business requirements.

These capabilities can be valuable for form-specific requirements such as:

- Additional validation;
- Calculated or derived values;
- Conditional behavior;
- Interaction between fields;
- Presentation adjustments;
- Lightweight extensions to the standard form experience.

Custom scripting should nevertheless be used with appropriate consideration for the long-term lifecycle of the solution.

As the amount and complexity of custom code increase, the solution can become increasingly dependent on the implementation of a particular eForms version or underlying technology.

Particular care should be taken where:

- Substantial business logic is implemented in client-side JavaScript;
- Many components depend on interconnected custom scripts;
- Scripts manipulate underlying form structures or implementation details;
- Custom code relies on behaviour beyond documented functionality;
- Significant application state is maintained within the form;
- Critical business functions depend on client-side customizations.

The fact that a particular requirement can technically be implemented through scripting does not necessarily mean that scripting represents the best long-term solution architecture.

Our recommendation is therefore to keep eForms customisation focused, well documented and closely related to the form experience.

Maintainability and product upgrades

Therefore™ continuously develops and enhances its products, including eForms.

Product evolution may include changes to components, underlying technologies and user-interface behavior. Consequently, custom code that depends on implementation details may need to be reviewed and, where necessary, adapted when moving between product versions.

For this reason, customer-developed scripts and other customizations should be included in normal solution testing and upgrade procedures.

In particular, **compatibility of customer-specific scripts that depend on implementation details cannot be guaranteed in the same way as documented standard product functionality.**

This does not mean that scripting should be avoided.

Rather, it means that partners and customers should make an architectural distinction between:

Standard product functionality

and

customer-specific code built on top of that functionality.

The more business-critical a solution becomes, the more important this distinction becomes.

A useful architectural principle is:

Keep form-specific behavior within the form, while keeping substantial or business-critical application logic independent from the implementation of the form wherever practical.

This approach reduces dependencies and helps make solutions easier to maintain and evolve.

Support Boundaries

Therefore provides product support for supported Therefore™ functionality and supported product configurations.

Where an issue relates to standard eForms functionality, it can be investigated through the normal Therefore™ support process.

Customer-specific JavaScript, CSS and other custom application logic must, however, be distinguished from the Therefore™ product itself.

The development, debugging and maintenance of customer-specific scripts are not part of standard Therefore™ product support.

This is consistent with existing Therefore™ support guidance, which distinguishes custom scripting from standard support activities.

Partners and customers are therefore responsible for the lifecycle of their customizations, including:

- Solution design;
- Quality and maintainability of custom code;
- Documentation of customizations;
- Testing their custom code;
- Maintaining customer-specific scripts;
- Validating customizations when upgrading;
- Adapting custom code where changes in the surrounding environment require this.

Therefore™ Support can investigate whether an observed issue originates in standard product functionality. The development, troubleshooting or adaptation of application-specific custom code remains the responsibility of the organization responsible for that customization.

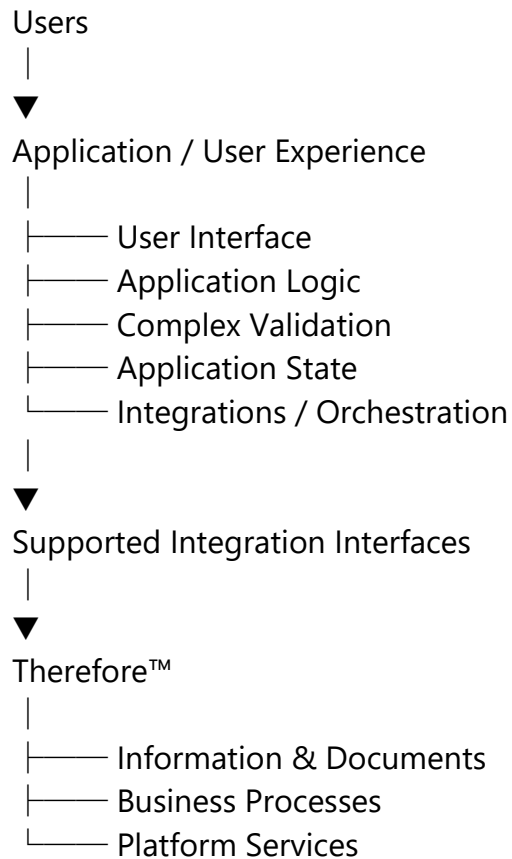
This separation provides a clear and sustainable distinction between **supporting the Therefore™ product** and **maintaining individually developed software**.

Recommended architecture for application-like requirements

When requirements extend significantly beyond form-based data capture, we recommend separating the **application layer** from the **Therefore™ platform**.

In this architecture, an external application provides the specialized user experience and application logic, while Therefore™ continues to provide the relevant information and process capabilities.

Conceptually:



The application layer can be developed using technologies and frameworks appropriate to the requirements of the individual solution.

This allows each part of the architecture to focus on its intended responsibility. The application can provide specialized user-interface behavior and application-specific logic, while Therefore provides the relevant information and process capabilities.

The applicable integration approach should use documented, supported interfaces for the relevant Therefore™ product and version.

Why we recommend this separation

For sophisticated solutions, separating substantial application logic from eForms provides several architectural benefits.

Clear responsibilities

The application layer is responsible for application-specific user experience and logic, while Therefore™ provides the information and process capabilities for which it is being used.

Better maintainability

Complex application logic can be developed and maintained using tools, frameworks and software-development practices specifically designed for application development.

Reduced coupling

Keeping substantial business logic outside individual forms reduces dependency on particular implementation details of eForms.

More flexible user experiences

Dedicated applications can evolve their user interfaces independently where specialized application experiences are required.

More predictable upgrade planning

Reducing implementation-specific dependencies can reduce the amount of custom form code that needs to be reviewed and adapted as the overall solution evolves.

Architectural flexibility

Different applications and user experiences can interact with the relevant Therefore™ capabilities without requiring every interaction to be implemented as an eForm.

Choosing the right approach

A simple set of architectural questions can help partners determine the appropriate technology.

Therefore™ eForms is a good fit for:	Consider a dedicated application when:
Users entering structured information	A substantial amount of JavaScript and business logic is being used
Validating information	Complex navigation is implemented and a highly dynamic user interface is required
Conditional use of fields	The application state must be maintained across interactions
Pre-populating and calculating field values	Several external service integrations are being used
Collecting documents and supporting information	Business-critical functions are implemented predominantly through client-side code
Initiating / Continuing a process with submission of a form	Strong reliance on implementation-specific form behavior.

The decision should therefore not be based simply on whether eForms is technically capable of implementing a requirement.

Instead, partners should consider:

Which architecture provides the clearest ownership, best maintainability and most sustainable solution over its expected lifecycle?

Recommended design principles

When designing solutions around Therefore™ eForms, we recommend the following principles:

1. Use eForms primarily for form-based data capture.

Keep the form focused on collecting, presenting and validating information.

2. Keep scripting proportional to the form requirement.

JavaScript and other customisation can be valuable, but extensive custom code should trigger an architectural review.

3. Avoid placing substantial business logic exclusively in the client.

Where business logic becomes central to the application, consider moving it to an appropriate application or service layer.

4. Minimise dependencies on implementation details.

Prefer documented functionality and supported interfaces wherever possible.

5. Design for the complete solution lifecycle.

Consider not only whether something can be implemented today, but also how it will be maintained, tested and upgraded.

6. Separate application development from form customisation when appropriate.

For full application experiences, use technologies designed for application development and integrate those applications with Therefore™ using documented, supported interfaces.

Summary

Therefore eForms remains a flexible and important part of the Therefore™ platform, providing powerful capabilities for electronic data capture and form-driven interaction with Therefore™.

Partners and customers are encouraged to make full use of these capabilities for appropriate form-based scenarios.

At the same time, we recommend maintaining a clear architectural distinction between an **electronic form** and a **business application**.

Use Therefore[™] eForms for form-based data capture, validation and lightweight form-specific logic.

Use Therefore[™] for the relevant information and business processes at the center of the solution.

Where requirements evolve into a full business application, implement the specialized user experience and substantial application logic using an appropriate application architecture and integrate it with Therefore through documented, supported interfaces.