



The European Standard for Fund Data Exchange

A Practical Handbook for Fund Professionals

Based on FundsXML Schema Version 4.2.x

2026

A Practical Handbook for Fund Professionals

FundsXML – The European Standard for Fund Data Exchange

Based on FundsXML Schema Version 4.2.x · 2026

Total length (including preface, appendices, and index): approx. 523 pages

Preface

Introduction to the book, target audience, structure, and the fictional "Europa Growth Fund" as the common thread throughout all chapters.

pages 1–4

Management Summary

A two-minute orientation for fund professionals

Executive overview of FundsXML: what problem it solves, what the standard covers, who benefits, and how this book is organised for different audiences. Front matter — recommended reading before Chapter 1.

pages 5–6

Part I — Foundations

Chapter 1 — Challenges in Fund Data Exchange

Why the industry needs a standard

Analysis of the data landscape: master, price, portfolio, and transaction data; data volumes and frequencies; data flows between asset managers, custodians, distributors, data vendors, and regulators. Typical problems without standardization, regulatory pressure as a driver, and an overview of existing standards.

pages 7–31

Chapter 2 — XML and XSD: The Technological Basis

Foundational knowledge for working with FundsXML

Concise introduction to XML and XSD: syntax, elements, attributes, namespaces, data types, validation, XSLT transformations, and an overview of key tools for day-to-day XML work.

pages 32–61

Chapter 3 — FundsXML: The Standard at a Glance

History, organization, and architecture

History since 2001, the organization and community, design principles, and the underlying schema architecture with its main areas (ControlData, Funds, AssetMasterData, Documents, RegulatoryReportings).

pages 62–81

Part II — FundsXML in Detail

Chapter 4 — ControlData: The Metadata of a Delivery

Document ID, reporting date, sender, and operation type

Structure and mandatory fields of the ControlData section, data sender and receiver, DataOperation types (INITIAL, UPDATE, DELETE), versioning, and a complete, schema-valid ControlData block as a practical example.

pages 82–103

Chapter 5 — Funds, Sub-Funds, and Share Classes

Static and dynamic fund information

Comprehensive presentation of the Fund structure: identifiers (ISIN, WKN, LEI, Valor), multilingual names, fund classification, share classes, and dynamic fund data (FundDynamicData). Full practical example based on the "Europa Growth Fund".

pages 104–143

Chapter 6 — Portfolio and Positions

Holdings and asset classes

Structure of the Portfolio section, all asset classes (Equity, Bond, Fund, Derivative, Cash, Real Estate, etc.), common data fields per position, linkage to AssetMasterData via UniqueIDs, and portfolio breakdowns. A mixed portfolio serves as an end-to-end practical example.

pages 144–188

Chapter 7 — Transactions and Orders

Orders, settlements, and corporate actions

Transaction types in FundsXML: subscriptions, redemptions, switches, distributions, income, and Order Execution Types. Rounded off by a complete practical example of a transaction file.

pages 189–218

Chapter 8 — Regulatory Modules

EMT, EPT, EET, EFT, and TPT in FundsXML

Embedding of the FinDatEx templates in FundsXML: European MiFID Template (EMT), European PRIIPs Template (EPT), European ESG Template (EET), European Feedback Template (EFT), and Tripartite Template (TPT) for Solvency II. Complemented by ESAP and a practical example of a complete regulatory delivery.

pages 219–268

Chapter 9 — Advanced Schema Areas

Factsheets, digital signatures, and CustomDataFields

Less frequently used but important schema areas: factsheet data and the Documents section, digital signatures based on XMLDSig, CustomDataFields for proprietary extensions, and country-specific additions.

pages 269–293

Part III — Implementation and Practice

Chapter 10 — Validation and Quality Assurance

Ensuring correct FundsXML files

The two-stage validation model (schema validation plus business validation), common validation errors and their resolution, automated quality checks (Schematron, business rules), and a complete validation workflow.

pages 294–318

Chapter 11 — Tools and Toolchain

The right tools for productive use

FreeXmlToolkit in detail, Online Schema Viewer, FundsXML CSV Converter, FundsXML Generator, and integration into development environments (IntelliJ, VS Code, Eclipse).

pages 319–346

Chapter 12 — FundsXML in the System Landscape

Integration into existing IT architectures

Typical architecture scenarios, comparison of programming languages (Java, Python, C#, JavaScript), reading and processing of FundsXML, database integration, data warehousing, and automation and scheduling.

pages 347–378

Chapter 13 — An Implementation Project from A to Z

A practical guide for introducing FundsXML

End-to-end walkthrough of a FundsXML project: project preparation and requirements analysis, mapping of existing data, prototyping and piloting, testing, acceptance and go-live, operations and maintenance.

pages 379–408

Part IV — Outlook and Reference

Chapter 14 — Future and Further Development

Where is FundsXML heading?

Current development directions, the role of FundsXML in an API-driven world, the influence of AI and automation, and opportunities to contribute to its further development.

pages 409–426

Appendices

Appendix A — Glossary

Technical terms in German/English with definitions and cross-references.

pages 427–438

Appendix B — XML Quick Reference

A compact cheat sheet for XML syntax, XPath, and XSLT.

pages 439–446

Appendix C — FundsXML Schema Overview

Structured overview of all important elements, types, and enumerations of the FundsXML schema.

pages 447–461

Appendix D — Complete Example Files

Complete, schema-valid FundsXML sample files for the "Europa Growth Fund".

pages 462–481

Appendix E — Resources and Links

Official sources, GitHub repository, community resources, and training offerings.

pages 482–485

Appendix F — Solutions to the Exercises

Model solutions for all chapter-level exercises.

pages 486–505

Index

Alphabetical index of key terms, elements, and concepts.

pages 506–515

FundsXML – The European Standard for Fund Data Exchange · © 2026



FRONT MATTER · MANAGEMENT SUMMARY

Management Summary

A two-minute orientation for fund professionals

The European fund industry moves more money than any other financial sector in the region, yet it spends an out-sized share of its operating budget simply **moving information about that money from one system to another**. Every asset manager, every administrator, every depositary, every distributor, every regulator consumes overlapping views of the same underlying facts — net asset values, holdings, costs, risks, sustainability disclosures — and for most of the industry's history, almost every pair of counterparties has invented its own way of exchanging them.

This book is about the European answer to that problem: **FundsXML**.

Why a standard is needed

A mid-sized UCITS distributed across ten European countries will today feed dozens of downstream systems — fund accounting, portfolio management, depositary oversight, data vendors, regulatory portals, distributor databases, ESG analytics — each expecting fund data in its own preferred format. The information is essentially identical; the packaging is not. One counterparty wants a bespoke CSV with forty columns; another insists on an Excel workbook; a third accepts only SWIFT messages; a fourth has built a proprietary XML dialect. The operations team is therefore obliged to maintain tens of bilateral interfaces to ship the same data in different shapes, and every small change at the source — a new ISIN, a reclassified cost component, a renamed benchmark — triggers work across every one of them.

Add to this the rising tide of regulatory templates — PRIIPs KIDs, EMT, EPT, EET, EFT, TPT, AIFMD Annex IV, MMF, and ESAP submissions — and the data-exchange problem becomes not a technical inconvenience but a **material component of the industry's cost base**. Standardisation, once a voluntary ambition, has become a commercial and regulatory necessity.

What FundsXML is

FundsXML is an **open, XML-based data standard for the exchange of fund-related information across the European asset-management industry**. Governed by a non-profit initiative and continuously developed since 2001, its current version — 4.2.x — covers the full spectrum of fund data that practitioners exchange every day:

- **ControlData** — who sent what, when, and with which operational intent;
- **Fund structure** — funds, sub-funds, and share classes, with their identifiers, descriptions, classifications, and fee structures;

- **Portfolio** — positions across all asset classes, linked to centralised asset master data;
- **Transactions and orders** — subscriptions, redemptions, switches, distributions, and corporate actions;
- **Regulatory modules** — the FinDatEx templates (EMT, EPT, EET, EFT, TPT) embedded natively in the schema;
- **Documents, signatures, and proprietary extensions** — factsheets, XMLDSig signatures, and CustomDataFields.

The standard is defined by a single XML Schema Definition (XSD), validated by open tools, and supported by a growing ecosystem of converters, libraries, and worked examples. FundsXML does not attempt to replace ISO 20022, SWIFT, FIX, or openfunds; it sits alongside them, focused on the *fund-data payload* that those transport standards were never designed to model in detail.

Who benefits

FundsXML is used in production by **asset managers, fund administrators, depositaries, distributors, data vendors, and regulators** across the European market, with particularly strong adoption in the German-speaking countries and in the Luxembourg and Irish cross-border fund centres. A mid-sized administrator that adopts FundsXML typically collapses several bilateral feeds into a single canonical delivery; an asset manager that accepts FundsXML as an input format reduces its onboarding time for new distributors from weeks to days; a regulator that consumes FundsXML receives the same data that the industry already produces internally, rather than requiring a parallel reporting chain.

The economic case is straightforward: **one format, one validation, one change process, many consumers.**

What this book delivers

The book is organised in four parts. **Part I — Foundations** sets out the data problem, the underlying XML and XSD technology, and the history and architecture of FundsXML. **Part II — FundsXML in Detail** examines each area of the schema — ControlData, Funds, Portfolio, Transactions, Regulatory Modules, and Advanced Areas — with complete, schema-valid worked examples. **Part III — Implementation and Practice** covers validation, tooling, system integration, and the end-to-end delivery of a FundsXML project. **Part IV — Outlook and Reference** looks ahead to FundsXML in an API-driven, AI-assisted world.

A fictional Luxembourg-domiciled UCITS — the **Europa Growth Fund** — runs through every chapter as a continuous narrative thread, so that abstract schema constructs are always illustrated against a concrete, realistic case.

Readers with operational or managerial responsibilities may read Parts I and III and dip into Part II as required. Readers implementing FundsXML — whether on the generating or the consuming side — will find Parts II and III the working core of the book. Readers new to XML itself should begin with Chapter 2 before moving on.



PART I — FOUNDATIONS · CHAPTER 1

Challenges in Fund Data Exchange

Why the industry needs a standard

1.1 Setting the Scene: A Month-End at the Europa Growth Fund

It is the last business day of the month. At the fictional — but entirely representative — **Europa Growth Fund**, a mid-sized UCITS equity fund domiciled in Luxembourg and distributed across eleven European countries, the monthly reporting cycle has just begun.

By 7:00 a.m. CET, the fund accounting system at the administrator has produced a provisional Net Asset Value. By 9:30 a.m., the portfolio management system at the asset manager has reconciled its shadow NAV against the official one. By 10:00 a.m., twenty-three downstream systems — ranging from the depositary's oversight platform to a data vendor's classification engine, from three regulators' reporting portals to eleven distributors' fact-sheet databases — expect a fresh delivery of fund data in their preferred format.

The operations team at the Europa Growth Fund knows this moment well. One distributor wants a proprietary CSV file with forty-seven columns in a specific German locale. Another insists on an Excel workbook with macros. A third accepts only a SWIFT MT535 message for holdings. A fourth has recently migrated to an in-house XML schema that — despite its name — bears little resemblance to any public standard. The German regulator needs a PRIIPs KID, the French regulator a DICI, the Austrian distributor an EMT file, and the Italian insurer a Tripartite Template for Solvency II. The ESG team in Paris is meanwhile waiting for an EET so that Article 8 disclosures can be updated on the fund's website by noon.

Every one of these deliveries contains, at its heart, **the same underlying data**: one fund, three share classes, one portfolio, one NAV, one set of costs, one set of risks. Yet the operations team maintains more than thirty bilateral interfaces to ship this same information into thirty different shapes. When a single field changes — a new ISIN is added, a benchmark is renamed, a cost component is reclassified — thirty mappings must be touched, thirty test cycles must be run, thirty sign-offs must be collected.

This chapter is about why that reality exists, what it costs, and what the fund industry has been trying to do about it. It is the motivational chapter of this book. No XML syntax appears in these pages; the technology begins in Chapter 2. Here, we build the case for a standard — and, ultimately, for **FundsXML** — by looking honestly at the data landscape that fund professionals navigate every day.

By the end of the chapter, you should be able to:

- describe the four principal categories of data that flow through the European fund industry;
- name the main actors in a typical fund data network and the flows that connect them;
- identify the recurring problems that emerge when these flows are handled without a common standard;

- place the most important competing and complementary standards — FIX, SWIFT, ISO 20022, openfunds, and FundsXML — on a single conceptual map;
- explain why regulatory pressure, rather than voluntary coordination, has become the dominant driver of standardisation in this decade.

1.2 The European Fund Industry in Numbers

To appreciate the data problem, it helps to appreciate the scale of the industry that produces it.

At the time of writing, European investment funds manage assets of roughly **twenty trillion euros**, split approximately two-thirds into UCITS (retail-oriented, passportable across the European Union) and one-third into AIFs (alternative investment funds, from hedge funds and private equity to real estate and infrastructure vehicles). Europe hosts more than **sixty thousand** individual funds and sub-funds; if one counts share classes separately, the figure comfortably exceeds **two hundred thousand** distinct investable units, each with its own ISIN.

Luxembourg and Ireland together domicile the majority of cross-border UCITS. France, Germany, the United Kingdom, Switzerland, and the Netherlands host large domestic markets in addition. Cross-border distribution — a UCITS sold in a country other than its country of domicile — is the norm rather than the exception, and each new distribution country typically adds its own reporting quirks.

The numerical profile of daily data traffic is sobering:

- Each open-ended fund produces **at least one NAV per business day**; many produce several (for different valuation points, currencies, or hedged share classes). That alone yields hundreds of thousands of NAV data points per day across Europe.
- **Portfolio** snapshots are typically reported **monthly** to the public and **daily** internally; a mid-sized equity fund holds 80 to 150 positions, while a global multi-asset fund may hold several thousand.
- Transactions — subscriptions, redemptions, switches, distributions, corporate actions — flow **continuously** throughout the trading day.
- Regulatory deliveries (EMT, EPT, EET, TPT, AIFMD Annex IV, MMF, PRIIPs) vary from **monthly** to **quarterly** to **ad-hoc**, with the direction of travel firmly towards higher frequency and broader scope.

Behind these numbers sits a simple observation: the European fund industry has become an **information-processing industry**. The cost of producing, distributing, reconciling, and reporting fund data is no longer a rounding error in the total expense ratio — it is a material component of the operating cost base of every asset manager, every administrator, every distributor, and every regulator. Standardisation is therefore not a purely technical concern; it is an industrial-economics concern.

The remainder of this chapter unpacks that observation in four steps. First, we classify the data itself. Second, we identify the actors that exchange it. Third, we examine what goes wrong when that exchange happens without a shared standard. Fourth, we survey the standards that have emerged in response — culminating in the positioning of FundsXML within that landscape.

1.3 What Flows? A Taxonomy of Fund Data

Fund data can be organised along many dimensions: by lifecycle (static versus dynamic), by sensitivity (public versus confidential), by origin (produced internally versus sourced externally). For the purposes of this book, we adopt a pragmatic four-part taxonomy that mirrors how FundsXML itself is structured and how most practitioners think about their day-to-day work:

1. **Master data** — the relatively stable descriptive information about funds, share classes, and instruments;
2. **Price data** — NAVs, market prices, and the time series built from them;
3. **Portfolio data** — what a fund actually holds at a given valuation point;
4. **Transaction data** — the events that change holdings and capital.

Each category has its own frequency, its own sources, its own consumers, and its own pathologies. We look at them in turn, with the Europa Growth Fund as a running example.

1.3.1 Master Data

Master data is the "identity layer" of the fund industry. It answers the questions *what is this fund*, *what is this share class*, *what is this instrument*. Typical master-data elements include:

- **Fund-level:** legal name, marketing name, LEI, inception date, fund type (UCITS, AIF, ETF), domicile, regulator, fund manager, administrator, depositary, auditor, fiscal year-end, base currency, benchmark.
- **Share-class-level:** ISIN, WKN, Valor, local identifiers, distribution policy (accumulating vs. distributing), currency, hedging, minimum investment, management fee, performance fee, TER, launch date, share-class-specific benchmark.
- **Instrument-level:** ISIN, CFI, issuer LEI, maturity, coupon, sector classification, country of risk, rating.

For the Europa Growth Fund, the master data fits on a handful of pages. Yet those pages are the **most-consulted and most-duplicated** information in the entire system landscape. Every downstream consumer — from a fact-sheet engine to a distribution platform to a ratings provider — needs a copy. Every consumer holds that copy in its own schema, subject to its own update cadence.

Master data changes rarely — perhaps a few dozen events per year for a typical fund. But when it does change, the ripple effects are enormous: a renamed benchmark, a new share class, a fee reduction, a change of depositary. Each such event must be propagated to every downstream copy, and each missed update is a latent reconciliation break.

The characteristic failure mode of master data is **silent divergence**: two systems hold "the same" fund under two slightly different names, two slightly different fee structures, two slightly different inception dates — and nobody notices until an auditor or a regulator asks a question that depends on precisely the field that has drifted.

1.3.2 Price Data

Price data is the most visible output of a fund. The NAV per share is the number investors see on their statements, the number published on websites, the number used to calculate performance fees, the number that drives subscription and redemption proceeds. For a daily-priced UCITS, the NAV is produced every business day after market close, validated overnight, and released the next morning.

Price data has several subtypes:

- **Official NAV** — the one used for subscriptions and redemptions; produced by the administrator, validated by the asset manager and depositary.
- **Indicative / intra-day NAV** — for ETFs, published throughout the trading day.
- **Shadow NAV** — the asset manager's own calculation for internal reconciliation.
- **Historical NAV time series** — used for performance calculations, risk indicators, and back-testing.
- **Dividend and distribution amounts** — technically an event rather than a price, but usually delivered together.
- **Performance indicators** — cumulative and annualised returns, ratios such as Sharpe and Sortino, volatility, tracking error.

Compared with master data, price data is **high-volume and highly time-sensitive**. A delayed NAV delays every downstream process that depends on it, from the publication of daily prices on Morningstar to the valuation of a feeder fund that holds the Europa Growth Fund as a position.

The characteristic failure mode of price data is **format fragmentation**. The NAV itself is a single decimal number, but the way it travels differs: one distributor takes a CSV file named `NAV_YYYYMMDD.csv`, another a SWIFT MT535 with an embedded price block, another a proprietary JSON endpoint, another an emailed Excel sheet. Multiply by share classes and currencies and the number of outbound NAV feeds for a mid-sized fund house easily exceeds one hundred per day.

1.3.3 Portfolio Data

Portfolio data describes, at a specific valuation point, everything a fund holds: securities, cash, derivatives, other funds, real assets, receivables, and payables. A single portfolio snapshot for the Europa Growth Fund might contain:

- one hundred and twenty equity positions in twenty-two countries;
- a basket of fifteen futures used to equitise cash;
- cash balances in five currencies;
- three FX forwards hedging the share-class-currency exposure;
- accrued dividends, pending trades, and management-fee accruals.

Each position has its own descriptive fields — identifier, description, quantity, price, market value, weight, currency, country of risk, sector, credit rating, maturity — and each consumer of the portfolio has its own requirements about which of those fields must be present, in what precision, and in which reference currency.

Portfolio data feeds:

- **risk systems**, which compute exposures, VaR, stress tests, and counterparty concentrations;
- **performance-attribution systems**, which decompose returns into allocation and selection effects;
- **compliance systems**, which check UCITS eligibility, concentration limits, and ESG exclusions;
- **regulatory reporting**, from AIFMD Annex IV to Solvency II look-through to PRIIPs risk calculations;
- **transparency reporting**, such as monthly top-ten holdings or full portfolio disclosures.

The characteristic failure mode of portfolio data is **semantic ambiguity**. The phrase "market value" has half a dozen legitimate interpretations (clean vs. dirty price for bonds, pre- vs. post-accrual, including vs. excluding cash). The phrase "country" may mean country of incorporation, country of tax residence, country of risk, or country of the primary listing. Without a shared glossary — and without a schema that forces the producer to declare which variant is intended — each consumer is free to guess, and guesses diverge.

1.3.4 Transaction Data

Transactions are the events that change a fund's holdings or its capital base. They come in two broad flavours:

- **Capital transactions** — subscriptions, redemptions, switches between share classes, distributions to investors;
- **Investment transactions** — buys, sells, coupon receipts, dividend receipts, corporate actions, collateral movements, FX trades.

Each transaction has a richer event-like structure than a static position: it has a trade date, a value date, a settlement date, a counterparty, a price, a commission, a tax, and — crucially — a **state** (pending, matched, settled, failed, cancelled). Over the life of a single order, multiple transaction records may be produced as the state evolves.

For the Europa Growth Fund, transaction data flows in two directions. Outbound transactions leave the portfolio management system for the administrator, the depositary, the brokers, and the clearing system. Inbound transactions arrive from the transfer agent (as share-class subscriptions and redemptions), from the custodian (as settlement confirmations), and from data vendors (as corporate-action notifications).

The characteristic failure mode of transaction data is **mismatched state**: two systems agree that a trade happened, but disagree about whether it is settled, whether the tax was withheld, or whether a corporate action has been applied. Reconciliation teams at asset managers spend a significant portion of their working day chasing exactly these mismatches.

1.3.5 A Consolidated View

Table 1.1 summarises the four categories along the dimensions most relevant for data-exchange design.

Table 1.1 — The four categories of fund data

Category	Typical frequency	Primary producer	Primary consumers	Dominant pain point
Master data	Rare events (days/weeks)	Asset manager / administrator	Distributors, vendors, regulators, website	Silent divergence between copies
Price data	Daily (intra-day for ETFs)	Administrator	Distributors, vendors, platforms, investors	Format fragmentation per channel
Portfolio data	Daily internal / monthly public	Administrator / portfolio manager	Risk, compliance, regulators, transparency feeds	Semantic ambiguity of fields
Transaction data	Continuous (event-driven)	Portfolio manager, transfer agent, custodian	Reconciliation, accounting, regulators	Mismatched state across systems

The insight to carry forward is that a **single standard must accommodate all four categories** if it is to meaningfully reduce the bilateral-mapping problem. A standard that covers only master data leaves the NAV and portfolio flows untouched; a standard that covers only transactions does nothing for the fact-sheet engine. The ambition of FundsXML — as we will see in Chapter 3 — is precisely to provide a single model that can express all four.

1.4 Who Exchanges Data with Whom? The Actors and Their Flows

If the data taxonomy is the *what*, the actor landscape is the *who*. A realistic picture of that landscape is the first step towards understanding why bilateral mappings multiply so quickly.

1.4.1 The Principal Actors

Around any European investment fund, one finds — at a minimum — the following roles:

- **The asset manager** (management company, KVG, ManCo) is the legal entity responsible for managing the fund. It makes investment decisions, maintains the investment strategy, and bears ultimate responsibility for the product.
- **The fund administrator** performs fund accounting, calculates the NAV, maintains the shareholder register (unless the transfer agent is a separate entity), and produces financial statements. In Luxembourg and Ireland, fund administration is typically outsourced to large specialised banks.
- **The depositary (custodian bank)** holds the fund's assets in safekeeping, monitors cash flows, and performs oversight duties mandated by UCITS and AIFMD.
- **The transfer agent** processes subscriptions, redemptions, and share-class switches on behalf of investors; it maintains the register of holders.

- **Distributors** — retail banks, private banks, independent financial advisers, online platforms, insurance companies, pension schemes — bring the fund to end investors. Each distributor has its own onboarding process, its own data requirements, and its own delivery preferences.
- **Data vendors and aggregators** — Bloomberg, Refinitiv, Morningstar, WM Datenservice, FundConnect, Kneip, MountainView, and many others — collect fund data from producers and redistribute it, in normalised form, to subscribers across the financial industry.
- **Regulators** — national authorities such as BaFin (Germany), AMF (France), CSSF (Luxembourg), Central Bank of Ireland, FMA (Austria), FINMA (Switzerland), as well as the European Securities and Markets Authority (ESMA) — require regular reporting under UCITS, AIFMD, MiFID II, PRIIPs, SFDR, and other regimes.
- **Auditors** review the fund's financial statements and regulatory disclosures annually.
- **Internal stakeholders** — the portfolio managers themselves, risk, compliance, product, legal, marketing, and client reporting teams — consume fund data for purposes ranging from daily decision-making to glossy brochures.

1.4.2 A Representative Flow Diagram

Figure 1.1 sketches the data-flow network of the Europa Growth Fund. The fund sits at the centre; around it are the actors just listed; between them run the arrows that represent data flows.

Figure 1.1 — Data flows around the Europa Growth Fund

(A full-page diagram placed here in the final book. The diagram shows the Europa Growth Fund at the centre, surrounded by the asset manager, administrator, depositary, transfer agent, distributors, data vendors, regulators, and internal consumers. Arrows are colour-coded by data category — master, price, portfolio, transaction — and labelled with typical frequencies.)

Even for a single mid-sized fund, the diagram has more than twenty arrows. For a fund house running a hundred funds, the number of bilateral flows runs into the thousands. It is this **combinatorial explosion** — not the volume of any single flow — that makes unstandardised data exchange so expensive.

1.4.3 Three End-to-End Flows in Detail

To make the picture concrete, consider three complete flows that together account for much of the daily traffic around the Europa Growth Fund.

Flow 1 — Daily NAV publication. At 18:00 CET, the administrator's fund-accounting system strikes the books. By 22:00, after overnight pricing feeds have been applied and the depositary has performed its cash-monitoring checks, a draft NAV is available. By 07:00 the next morning, the asset manager's oversight team reviews the NAV against its shadow calculation. On release, the NAV must be delivered to: the asset manager's website, the administrator's own transparency portal, Bloomberg, Refinitiv, Morningstar, three regional data vendors, eleven distributors, and two fund-of-funds that hold the Europa Growth Fund as a position. Each of these recipients expects the NAV in its preferred format.

Flow 2 — PRIIPs KID generation and distribution. Every twelve months, and whenever market conditions materially change, a Key Information Document must be produced for each share class of the Europa Growth Fund. The KID contains risk indicators, cost disclosures, and performance scenarios — all of which depend on portfolio data, price history, and cost data. Once generated, the KID PDF must be pushed to every distributor in every country where the fund is sold, in the local language, with version tracking that allows investors to retrieve the exact KID in effect on the day they subscribed. The data feeding the KID is the same data feeding half a dozen other reports; the delivery mechanism is entirely different.

Flow 3 — MiFID II cost and target-market reporting. Under MiFID II, distributors must receive, for every share class they sell, a European MiFID Template (EMT) detailing target market, costs, and charges. The EMT is refreshed monthly or quarterly and whenever the underlying data changes. It is distributed via a mix of FinDatEx channels, data-vendor repositories, and direct bilateral feeds. For a fund sold in eleven countries through a hundred distributors, dozens of EMT deliveries per month are not unusual.

The point of walking through these three flows is to emphasise that **the same underlying dataset feeds very different outbound products** — and that the effort of producing those outbound products, in the absence of a standard, scales linearly with the number of downstream formats rather than with the complexity of the data itself.

1.5 What Goes Wrong Without a Standard

Having mapped the data and the actors, we now turn to the recurring pathologies that afflict unstandardised data exchange. Each of the six pathologies below is common enough that every experienced fund-operations professional will recognise it. For each, we offer a short description, a concrete illustration from the Europa Growth Fund, and a sense of the cost it imposes.

1.5.1 Proprietary Formats and the n^2 Interface Problem

The most immediate symptom of missing standardisation is the proliferation of bilateral, proprietary formats. When n parties need to exchange data and no shared format exists, the number of distinct mappings required scales as $n \times (n - 1)$, i.e. roughly as the square of the number of participants. A network of ten counterparties requires ninety mappings; one hundred counterparties, roughly ten thousand.

At the Europa Growth Fund, the outbound NAV feed illustrates the problem in miniature. Thirteen distributors receive the daily NAV:

- four accept a CSV with semicolon separators and German decimal comma;
- three accept the same CSV with comma separators and English decimal point;
- two require an Excel workbook with a specific sheet name;
- two use a proprietary XML dialect inherited from a legacy project;
- one consumes a SWIFT MT535;
- one ingests a REST endpoint exposing JSON.

None of these formats is inherently wrong. Each was rational at the time it was chosen. But collectively, they mean that a change to the NAV payload — for example, adding a hedged-currency NAV for a new share class — triggers **thirteen distinct change requests**, thirteen distinct test cycles, and thirteen distinct go-live windows. The marginal cost of adding the fourteenth distributor is not the cost of understanding one more format; it is the cost of adding yet another line to an already-sprawling maintenance matrix.

A shared standard collapses $n \times (n - 1)$ into something closer to n : each participant implements the standard once, and bilateral mappings disappear. This is the single most important economic argument for standardisation — and it is, in microcosm, the argument for FundsXML.

1.5.2 Identifier Confusion

A fund or instrument rarely has just one identifier. It may have an ISIN (international), a WKN (German market), a Valor (Swiss market), a SEDOL (UK market), a Bloomberg ticker, a RIC (Refinitiv Instrument Code), a CUSIP (for North-American assets), an in-house code, and a legacy identifier inherited from a pre-merger system. Each system in the chain privileges one of these identifiers as primary and treats the others as aliases — if it tracks them at all.

At the Europa Growth Fund, a recurring incident pattern involves corporate actions on small-cap holdings. The custodian reports the event using the ISIN of the new security; the portfolio management system stores positions under WKN; the risk system uses an in-house code that maps to the pre-event ISIN. For three days after the event, reports from the three systems disagree, and the reconciliation team must manually trace the event through each identifier.

A common standard does not, by itself, eliminate the existence of multiple identifiers — the world has many identifiers for good reasons. But a standard can specify **how multiple identifiers are carried together** on the same record, and which is authoritative, so that the consumer need not guess. FundsXML makes this explicit through its `Identifiers` and `OtherIDs` structures, as we will see in Chapter 5.

1.5.3 Semantic Inconsistency

Two systems may use the same field name for subtly different concepts. A few examples:

- **NAV** may mean the official NAV, the indicative NAV, the gross NAV (before the performance fee crystallisation), the net NAV (after all fees), the pre-swing NAV, or the post-swing NAV.
- **Total Expense Ratio** may be calculated on an ex-ante basis or an ex-post basis, may include or exclude performance fees, may include or exclude transaction costs.
- **Country** may mean country of domicile, country of incorporation, country of risk, country of tax residence, country of the primary listing, or country of the underlying exposure.
- **Exposure** may be gross, net, commitment-based, or VaR-based.

At the Europa Growth Fund, an illustrative case involves the ESG team's monthly report. The team reports carbon intensity using country of risk; the marketing team's fact-sheet engine reports country allocation using country of incorporation; a third-party research provider uses country of listing. All three numbers end up on the fund's website, and none of them matches. An internal audit eventually determines that all three are technically correct — they just answer different questions. But investors and sales staff, unaware of the distinction, spend weeks trying to reconcile what cannot be reconciled.

A standard addresses semantic inconsistency through two mechanisms: a **typed schema** that constrains what a field can contain, and a **shared glossary** that defines what each field means. FundsXML provides both, and the habit of consulting the schema definition before using a field is perhaps the most important discipline a fund-data practitioner can acquire.

1.5.4 Versioning and Release Chaos

Every format evolves. A new regulatory requirement adds a field; a deprecated field is removed; a type is tightened; an enumeration gains new values. In an unstandardised environment, each bilateral format evolves independently, on its own schedule, with its own change-notification mechanism — if any.

At the Europa Growth Fund, the operations team has a wall-chart that tracks the effective version of each outbound format by counterparty. The chart has over sixty rows. When a counterparty announces a format change, the team must plan a migration window, produce parallel outputs during the transition, and coordinate cut-over with the receiver — all while maintaining the other fifty-nine flows unchanged. Mistakes manifest as delivery failures at precisely the moment no one can afford them: the month-end cycle.

A shared standard does not eliminate versioning — FundsXML itself has evolved from 1.0 to 4.2.x over more than two decades — but it concentrates the versioning discussion into a **single conversation** that all participants can follow, rather than hundreds of bilateral negotiations.

1.5.5 Manual Work and Operational Risk

The preceding four pathologies all share a common consequence: they move work from machines, which are cheap, to humans, who are expensive, slow, and error-prone. Every field that cannot be parsed automatically must be read by someone. Every reconciliation break must be investigated by someone. Every format migration must be project-managed by someone.

Industry benchmarks vary, but it is not unusual for a mid-sized fund house to estimate the fully-loaded cost of a single bilateral feed interface at **tens of thousands of euros per year** for maintenance alone, exclusive of the original build cost. For a fund house with five hundred active interfaces, the annual cost of *not* being standardised runs into the millions. This cost is invisible on any single balance sheet line, because it is spread across operations, IT, change management, and vendor contracts — but it is very real, and it is ultimately borne by the end investor in the form of management fees.

Beyond cost, there is **risk**. Every manual touch is an opportunity for error. Every reconciliation break, if missed, can turn into a pricing error, a regulatory finding, or a reputational incident. The fund industry has learned, through painful experience, that operational risk is not a separate category of risk; it is a first-class risk that can easily exceed market risk for some activities.

1.5.6 The Hidden Cost of Non-Standardisation

Pulling the five pathologies together yields a simple conclusion: the cost of *not* standardising is not zero, and it is not constant. It grows non-linearly with the number of participants, the number of formats, and the pace of regulatory change. For decades, the industry tolerated this cost because each individual participant perceived its own share as manageable. In the past decade, three things have changed:

- **Regulation** has multiplied the number of outbound data products (see Section 1.6), shifting the cost curve sharply upward.
- **Cross-border distribution** has multiplied the number of bilateral endpoints per fund.
- **Data-driven products** — robo-advisers, ETF platforms, ESG-labelled mandates — have created new consumers with new requirements, accelerating format proliferation.

Standardisation, in other words, used to be a nice-to-have. It is now an economic necessity. The question is no longer *whether* to standardise, but *which standard*, applied to *which part of the flow*. The rest of this chapter turns to that question.

1.6 Regulation as the Dominant Driver

If one had to pick a single reason why FundsXML — and standardisation in general — has moved from the periphery to the centre of fund operations in the past decade, it would be regulation. Voluntary industry coordination, historically, has been slow; regulatory deadlines have not.

1.6.1 The Regulatory Stack at a Glance

European fund regulation is dense. The list below is not exhaustive, but it covers the regulations that most directly drive data-exchange requirements:

- **UCITS** (Undertakings for Collective Investment in Transferable Securities) — the foundation of European retail fund regulation, mandating investment restrictions, risk management, and investor disclosure. UCITS IV introduced the Key Investor Information Document (KIID); UCITS V and VI tightened depositary rules and remuneration disclosures.
- **AIFMD** (Alternative Investment Fund Managers Directive) — the parallel regime for AIFs, with extensive regulatory reporting through Annex IV.
- **MiFID II / MiFIR** (Markets in Financial Instruments Directive/Regulation) — imposes target-market, cost, and charges disclosures on distributors, driving the EMT template.
- **PRIIPs** (Packaged Retail and Insurance-based Investment Products Regulation) — replaces the UCITS KIID with the PRIIPs KID for retail products, driving the EPT template for the underlying calculations.
- **SFDR** (Sustainable Finance Disclosure Regulation) and the **EU Taxonomy Regulation** — mandate sustainability-related disclosures at entity and product level, driving the EET template.
- **Solvency II** — requires insurance companies to look through their fund holdings to the underlying assets for capital calculations, driving the TPT template.
- **Money Market Fund Regulation** — specific disclosure and reporting rules for MMFs.
- **DORA** (Digital Operational Resilience Act) — elevates the operational-resilience expectations on all regulated financial entities, indirectly raising the bar for data-exchange quality.
- **ESAP** (European Single Access Point) — creates a central EU repository for public corporate and financial information, expected to become a major aggregation point for fund data in the coming years.

1.6.2 From Regulation to Data Requirement

Each of these regulations translates into concrete data requirements. Table 1.2 maps a selection of regulations onto the data categories they most directly affect and onto the FundsXML module that expresses them — a forward reference to Chapter 8, where the regulatory modules are treated in depth.

Table 1.2 — Regulation to FundsXML module

Regulation	Primary data required	FundsXML touchpoint
MiFID II (target market, costs)	Share-class master data, costs, target-market classification	EMT within RegulatoryReportings
PRIIPs KID	Risk indicator, cost table, performance scenarios, historical NAVs	EPT within RegulatoryReportings
SFDR / EU Taxonomy	ESG product classification, PAI indicators, taxonomy alignment	EET within RegulatoryReportings
Solvency II look-through	Full portfolio, issuer ratings, country of risk, durations	TPT within RegulatoryReportings
AIFMD Annex IV	Portfolio, leverage, liquidity, counterparty exposures	Portfolio + CustomDataFields
UCITS / MMFR disclosures	NAV, portfolio composition, liquidity buckets	FundDynamicData + Portfolio
ESAP	Consolidated public fund disclosures	Documents + regulatory modules

The practical consequence of this mapping is important: **regulators, although they do not mandate FundsXML directly, increasingly mandate the data that FundsXML is designed to carry.** The path of least resistance for most fund houses is to produce a single canonical FundsXML representation of their regulatory data and transform it into whatever filing format the regulator actually accepts. The canonical representation is what makes the regulatory workload manageable in the first place.

1.6.3 The Pace of Change

Ten years ago, a mid-sized fund house might have had one major regulatory data project per year. Today, it is rarely fewer than three concurrently: a PRIIPs refinement, an SFDR amendment, a MiFID II review, a Solvency II look-through change, a DORA implementation. The regulatory roadmap for the remainder of this decade shows no sign of slowing, and the direction of travel is unmistakable: **more fields, more frequently, more comparably, more publicly.**

Against that backdrop, the argument for a stable, well-governed, schema-based standard is no longer intellectual. It is operational. A fund house that enters each new regulatory project with a canonical data model already in place pays the fixed cost once; a fund house that treats each project as a bespoke initiative pays the fixed cost again and again.

1.7 The Standards Landscape

FundsXML is not alone. Several standards — some older, some newer, some complementary, some competing — populate the European fund data landscape. A practitioner reading this book should be able to place each of them on a mental map. This section provides that map.

1.7.1 FIX Protocol

The **Financial Information eXchange (FIX) protocol** was born in 1992 out of a bilateral effort between Fidelity Investments and Salomon Brothers to standardise equity-trade communications. Today, FIX is the dominant protocol for electronic order entry and trade execution in equity, derivatives, and increasingly fixed-income markets. Every sell-side trading desk speaks FIX; every order management system on the buy side speaks FIX; every execution venue exposes a FIX gateway.

FIX is a **session-oriented, message-oriented protocol**, optimised for low-latency, high-volume order flow. Its data model is centred on orders and executions, not on funds. It does not attempt to describe a fund's master data, its NAV, or its regulatory disclosures. It is, in the taxonomy of Section 1.3, primarily a **transaction-data standard** — and within transactions, primarily investment transactions rather than capital transactions.

FIX is therefore **complementary** to FundsXML, not competing. A fund's portfolio manager uses FIX to send orders to brokers; the resulting executions feed the portfolio that FundsXML then describes.

1.7.2 SWIFT MT and MX

SWIFT (Society for Worldwide Interbank Financial Telecommunication) operates the global messaging network that underpins interbank payments, securities settlement, trade finance, and treasury. Its legacy **MT message series** — MT5xx for securities — has been the backbone of cross-border custody and settlement for decades. MT535 carries statements of holdings; MT536 carries statements of transactions; MT540–MT548 carry settlement instructions and confirmations.

SWIFT is migrating its messages to **MX**, the XML-based successor built on ISO 20022. The migration is well under-way for payments and is progressing, more slowly, for securities.

For the fund industry, SWIFT is the dominant channel between **funds and their custodians and depositaries**. It handles holdings statements, settlement confirmations, income events, and corporate-action notifications. Like FIX, SWIFT is **complementary** to FundsXML: it covers a specific segment of the flow — the custody and settlement leg — that FundsXML does not attempt to address directly. A typical operational architecture consumes SWIFT messages from the custodian, translates them into internal representations, and re-emits the resulting fund data in FundsXML for onward distribution.

1.7.3 ISO 20022

ISO 20022 is the international standard for financial-industry messaging. Unlike FIX, SWIFT MT, or FundsXML, ISO 20022 is not a single message format; it is a **methodology** — a metamodel and a repository of business components — from which concrete messages in many domains are derived. Payments, securities, cards, foreign exchange, trade services, and increasingly investment funds all have ISO 20022 message sets.

For investment funds, ISO 20022 defines message families such as `setr` (order processing), `semt` (statements), `sese` (settlement), and `reda` (reference data). These messages overlap in scope with FIX, SWIFT MT, and — in the master-data and price areas — with FundsXML.

The relationship between ISO 20022 and FundsXML is subtle. ISO 20022 is a broad, cross-industry, cross-domain framework governed by a formal ISO process; its governance is heavyweight and its release cycles are long. FundsXML is a **focused, fund-industry-specific standard** that can iterate faster and cover fund-specific concepts — regulatory templates, FinDatEx embedding, multi-lingual fact-sheet text — that an ISO 20022 message would struggle to represent natively. The two standards are best thought of as operating at different levels of abstraction: ISO 20022 dominates interbank and custody flows; FundsXML dominates fund-product-centric flows. They coexist, and well-architected systems translate between them at the boundary.

1.7.4 openfunds

openfunds is a lightweight, community-driven initiative originating in Switzerland that defines a **standardised vocabulary of fund master-data fields**. Rather than prescribing a message format, openfunds publishes a catalogue of approximately five hundred field definitions — each with a stable identifier, a clear definition, a type, and guidance on usage. The catalogue has been widely adopted by Swiss and German market participants for the exchange of fund reference data.

openfunds answers the semantic-inconsistency problem of Section 1.5.3 for master data: when two parties agree to use openfunds field `OFST005010` for a benchmark name, they agree on what the field means. openfunds does not, however, prescribe how that field should be carried on the wire — whether as CSV, JSON, Excel, or XML — nor does it cover portfolios, transactions, or regulatory reporting.

openfunds is therefore **complementary** to FundsXML in theory and, in practice, has strongly influenced the master-data portions of the FundsXML schema. Many openfunds fields map directly to FundsXML elements, and a common pattern is to use openfunds identifiers as the authoritative semantic reference while using FundsXML as the transport and structural container.

1.7.5 FundsXML

FundsXML is the subject of this book. In one sentence: FundsXML is a comprehensive, schema-based XML standard for the exchange of **fund-centric data** — master data, price data, portfolio data, transactions, and regulatory reporting — governed by an industry initiative and maintained as an open standard.

Its defining characteristics, each of which we will revisit in depth in later chapters, are:

- **Fund-centric scope.** Unlike FIX (trade-centric), SWIFT (message-centric), or ISO 20022 (bank-centric), FundsXML is built around the concept of a fund and its share classes. A single FundsXML document can describe a fund's identity, its NAV, its holdings, its transactions, and its regulatory disclosures in one coherent

structure.

- **Schema-based validation.** FundsXML is defined by an XSD schema, enabling automated validation of every incoming and outgoing document. This single property eliminates entire categories of silent errors.
- **Regulatory embedding.** FundsXML provides first-class support for the FinDatEx templates (EMT, EPT, EET, EFT, TPT), allowing a single delivery to carry all regulatory data a distributor needs.
- **Extensibility.** The `CustomDataFields` mechanism allows proprietary extensions without breaking schema compatibility.
- **Open governance.** The standard is maintained by an industry initiative with representatives from asset managers, administrators, vendors, and regulators.

1.7.6 A Comparative Map

Table 1.3 places the five standards side by side along four dimensions: scope, message type, governance model, and depth of adoption in the European fund industry.

Table 1.3 — A comparative map of fund-industry standards

Standard	Primary scope	Message type	Governance	European fund adoption
FIX	Order entry, execution	Session-based, tag/value	FIX Trading Community	Ubiquitous for trading
SWIFT MT/MX	Custody, settlement, payments	Store-and-forward messages	SWIFT cooperative	Ubiquitous for custody
ISO 20022	Cross-industry financial messaging	XML messages from a metamodel	ISO / RMG	Growing, heterogeneous
openfunds	Master-data vocabulary	Field catalogue (format-agnostic)	openfunds association	Strong in DACH region
FundsXML	Fund product data end-to-end	Schema-based XML documents	FundsXML initiative	Strong and growing, especially for regulatory flows

The map highlights that FundsXML occupies a **distinct and largely uncontested niche**: the end-to-end description of a fund product — its identity, its dynamics, its holdings, its regulatory disclosures — as a single validated document. The other standards are not displaced; they continue to dominate their respective niches. A realistic operational architecture uses several of them together: FIX to the brokers, SWIFT to the custodian, openfunds as a semantic reference, FundsXML as the outbound distribution and regulatory medium.

1.8 Positioning FundsXML

The preceding sections have built, step by step, the case for a standard that:

- covers all four categories of fund data in a single coherent model;
- addresses the combinatorial explosion of bilateral mappings;
- embeds regulatory templates as first-class citizens;
- rests on a validated schema rather than on informal agreements;
- complements, rather than competes with, the established transaction-level and custody-level standards.

FundsXML is the standard that most completely satisfies these criteria for the European fund industry today. It is not the only answer to any single question; but it is the only standard that attempts to answer *all* of the questions together. Its closest conceptual neighbour, ISO 20022, is broader but less fund-specific and slower to evolve. Its closest semantic neighbour, openfunds, is sharper on master-data definitions but does not prescribe a transport structure. Its closest transactional neighbours, FIX and SWIFT, do not attempt to describe a fund product at all.

Viewed in this landscape, the choice for a fund-industry practitioner is not usually *FundsXML versus something else*, but rather *FundsXML together with the right neighbours* — with FIX or equivalent at the trading interface, SWIFT or ISO 20022 at the custody interface, openfunds as a shared dictionary, and FundsXML as the canonical fund-product representation that feeds distributors, regulators, and transparency channels.

For the Europa Growth Fund, the implication is concrete. By adopting FundsXML as the canonical internal model for fund-product data, the operations team can:

- retire bilateral proprietary formats in favour of a single outbound FundsXML stream that each consumer transforms locally;
- generate all FinDatEx regulatory templates from one authoritative source, eliminating cross-template inconsistency;
- validate every outbound document against a schema before it leaves the building, catching errors at the cheapest possible point;
- negotiate format changes with counterparties through a single version conversation rather than through thirty bilateral negotiations.

None of this happens overnight, and Chapter 13 devotes itself to the project mechanics of getting from today to that future state. But the destination is worth naming now, at the start of the book, so that every subsequent chapter can be read with a clear picture of what it contributes.

1.9 Key Takeaways

- The European fund industry is, in economic terms, an information-processing industry; the cost of moving fund data across the ecosystem is material and growing.
- Fund data divides naturally into four categories — master data, price data, portfolio data, and transaction data — each with its own frequency, producers, consumers, and characteristic failure modes.

- A realistic flow diagram around a single fund involves more than twenty bilateral exchanges; at the level of a fund house, the number runs into the thousands and grows as the square of the number of counterparties in the absence of a standard.
- The dominant pathologies of unstandardised exchange are proprietary-format proliferation, identifier confusion, semantic inconsistency, version chaos, and operational risk — all of which translate directly into cost and, ultimately, into fees borne by the end investor.
- Regulatory pressure has transformed standardisation from a nice-to-have into an operational necessity; a canonical fund-data model is now the path of least resistance through an ever-denser regulatory stack.
- FIX, SWIFT, ISO 20022, openfunds, and FundsXML coexist on a shared landscape. FundsXML occupies the distinct and largely uncontested niche of **end-to-end fund-product description**, and is complementary to, rather than competing with, the transaction-level and custody-level standards.
- The rest of this book develops, in detail, how FundsXML fulfils that role — starting in Chapter 2 with the XML and XSD technology on which it rests, and proceeding in Chapter 3 to the architecture of the standard itself.



PART I — FOUNDATIONS · CHAPTER 2

XML and XSD — The Technological Basis

Foundational knowledge for working with FundsXML

2.1 From "Why" to "How": A Short Hop

Chapter 1 built the case for a single, shared standard for fund data. It did so deliberately without a single line of XML: the argument there was economic and operational, not technical. This chapter begins the technical half of the journey. It introduces XML and XSD — the two technologies on which FundsXML rests — and it does so from the perspective of a fund professional who needs to *read*, *validate*, *query*, and *transform* fund-data documents, not from the perspective of a software engineer building an XML parser from scratch.

Return, for a moment, to the Europa Growth Fund. At the end of Chapter 1, the operations team had decided — at least in principle — to adopt FundsXML as the canonical representation of its fund-product data. The next question on the team's mind is very practical: *what does such a FundsXML file actually look like?* When a colleague drops a 2-megabyte document on the desk and says "this is the monthly delivery", what should we expect to see? How do we confirm that the file is correct before we send it out? How do we extract the twenty fields that the fact-sheet engine needs? How do we turn the same file into the CSV that a Swiss distributor still insists on?

Those are the questions this chapter answers in general terms. Chapter 3 will then apply the answers to the specific shape of the FundsXML schema; Chapters 4 to 9 will walk through the schema module by module; Chapter 10 will explain the two-stage validation discipline that keeps production deliveries healthy. Everything from here on depends on the vocabulary of XML and XSD being second nature, and so the investment of thirty pages at this point is repaid many times over in the chapters that follow.

The chapter is deliberately practical. XML is a large specification with many corners that FundsXML does not use, and this book does not visit them. Where a topic is genuinely optional for a fund practitioner, we say so and move on. Where a topic matters every day — element-versus-attribute design, schema validation, namespaces, the basic XPath idiom — we spend the time.

By the end of this chapter, you should be able to:

- read and write well-formed XML documents and recognise the most common syntactic mistakes;
- explain the purpose of XML namespaces and interpret a namespaced document;
- read an XSD schema and map its declarations to the instance documents they constrain;
- describe the difference between well-formedness and schema validity, and between schema validation and business-rule validation;
- navigate an XML document with elementary XPath expressions;
- understand what an XSLT transformation does and follow a simple stylesheet end to end;

- name the tools a fund-industry practitioner typically uses for day-to-day XML work and know which later chapter treats each of them in depth.

No prior XML experience is assumed. Readers who already write XSLT 3.0 in their sleep may skim the first half of the chapter and rejoin us at §2.10.

2.2 XML in One Page

At its heart, XML is a notation for writing down trees of data as plain text. A document is a single tree with a single root. Every branch of the tree is an *element*. Each element has a name, may carry *attributes*, may contain other elements, and may contain text. That is essentially the whole data model; everything else in this chapter is either syntax that expresses this model or a schema language that constrains it.

The Europa Growth Fund, stripped to its bare essentials, looks like this in real FundsXML:

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
    <DataOperation>INITIAL</DataOperation>
    <Language>en</Language>
  </ControlData>
  <Funds>
    <Fund>
      <Identifiers>
        <LEI>549300ABCDEFGH1234</LEI>
      </Identifiers>
      <Names>
        <OfficialName>Europa Growth Fund</OfficialName>
      </Names>
      <Currency>EUR</Currency>
      <SingleFundFlag>true</SingleFundFlag>
      <FundDynamicData>
        <TotalAssetValues>
          <TotalAssetValue>
            <NavDate>2026-03-31</NavDate>
            <TotalAssetNature>OFFICIAL</TotalAssetNature>
```

```

        <TotalNetAssetValue>
          <Amount ccy="EUR">464552848.78</Amount>
        </TotalNetAssetValue>
      </TotalAssetValue>
    </TotalAssetValues>
  </FundDynamicData>
  <SingleFund>
    <ShareClasses>
      <ShareClass>
        <Identifiers>
          <ISIN>LU1234567890</ISIN>
        </Identifiers>
        <Names>
          <OfficialName>Europa Growth Fund I EUR Acc</OfficialName>
        </Names>
        <Currency>EUR</Currency>
      </ShareClass>
      <ShareClass>
        <Identifiers>
          <ISIN>LU1234567892</ISIN>
        </Identifiers>
        <Names>
          <OfficialName>Europa Growth Fund R CHF Acc Hedged</OfficialName>
        </Names>
        <Currency>CHF</Currency>
      </ShareClass>
    </ShareClasses>
  </SingleFund>
</Fund>
</Funds>
</FundsXML4>

```

Three observations are worth making before going further. First, this is a minimal but **schema-valid** FundsXML document — it validates against the real `FundsXML4.xsd` without errors (Appendix D contains the full set of progressively richer examples). The document carries a `ControlData` envelope that identifies the delivery, followed by a `Funds` block with a single fund and its two share classes. Chapters 4 and 5 explain every element in this document in depth; for now, read it as a map of the territory ahead.

Second, the document is entirely human-readable. A fund operations analyst can inspect it with any text editor and, after a minute of orientation, understand it without tooling. The element names — `ControlData`, `Fund`, `Names`, `OfficialName`, `Currency`, `ShareClass`, `ISIN` — read almost like a structured form. This readability is one of XML's enduring strengths and is a large part of why regulators and industry bodies have chosen XML-based formats for durable disclosures.

Third, the tree structure is visible in the indentation but does not depend on it. XML ignores most whitespace between elements; the indentation is a convenience for humans. Parsers see the same tree regardless of how the file is formatted.

A note on the examples in this chapter. The document above and the share-class examples later in this chapter are real FundsXML validated against the 4.2.8 schema. A small number of examples in the XML-syntax and namespace sections (§2.3–§2.5) use simplified, hand-rolled elements to illustrate general XML concepts in isolation; where this is the case, the text says so. Every example that claims to be FundsXML is schema-valid.

With this much in hand, we can now be precise about the syntactic rules that make a document well-formed.

2.3 XML Syntax — Elements, Attributes, Text

2.3.1 Well-formedness

An XML document is *well-formed* if it obeys a small number of strict rules. A well-formed document can be parsed by any conforming XML processor, regardless of whether it matches any schema. The rules are:

- **Exactly one root element.** The outermost pair of tags encloses everything. In the Europa Growth Fund example above, `<FundsXML4>` is the root.
- **Every start tag has a matching end tag.** `<Name>Europa Growth Fund</Name>` is well-formed; `<Name>Europa Growth Fund` is not.
- **Elements nest properly.** `<a><b></b></a>` is well-formed; `<a><b></a></b>` is not.
- **Element and attribute names are case-sensitive.** `<Fund>` and `<fund>` are different elements.
- **Attribute values are always quoted.** Single or double quotes are both allowed: `id="EGF-001"` and `id='EGF-001'` are equivalent.
- **Five characters are reserved and must be escaped** when they appear as text: `<` becomes `<`; `>` becomes `>`; `&` becomes `&`; `"` becomes `"`; and `'` becomes `'`.

An element with no content can be written as `<ShareClass></ShareClass>` or, equivalently, as the empty-element form `<ShareClass/>`. Parsers treat the two forms as identical.

A well-formed document is not necessarily a *useful* document: it may still violate a schema, contain the wrong fields, or carry impossible values. Well-formedness is the bottom rung of the validation ladder, not the top.

2.3.2 Elements versus attributes

One of the oldest debates in XML design is the choice between expressing a piece of information as a child element or as an attribute. Consider two hypothetical ways of writing a fund's base currency (these are general XML illustrations, not FundsXML elements):

```
<!-- As a child element -->
<Fund>
  <BaseCurrency>EUR</BaseCurrency>
</Fund>
```

```
<!-- As an attribute -->
<Fund baseCurrency="EUR"/>
```

Both carry the same information. The working rules of thumb, applied consistently throughout FundsXML and throughout the examples in this book, are:

- **Use elements for data, attributes for metadata.** The currency of a fund is data; the version of a schema or the identifier of a document is metadata about the document itself.
- **Use elements when the value may later need sub-structure.** A fund name today is a string; tomorrow it might need a language tag or a sort form. Elements can grow children; attributes cannot.
- **Use attributes for values that qualify the enclosing element without being part of its content.** The `id` on a record, the unit of a measurement, the version of a format.
- **Prefer elements when in doubt.** FundsXML leans heavily towards elements, and the habit of reaching for an element first is the safer default for a practitioner.

Illustrated on a hypothetical share-class element (not the actual FundsXML element, which uses `Identifiers/ISIN` and `Names/OfficialName` instead):

```
<ShareClass id="EGF-001-ACC-EUR" currency="EUR">
  <ISIN>LU0000000001</ISIN>
  <Name xml:lang="en">Europa Growth Fund, Accumulating, EUR</Name>
  <Name xml:lang="de">Europa Growth Fund, Thesaurierend, EUR</Name>
</ShareClass>
```

Here `id` and `currency` are qualifiers — they tell us *which* share class and in *what unit* — while the `ISIN` and the two `Name` elements carry the share class's own data. Note how the element form of `Name` naturally accommodates two languages; an attribute form could not. The real FundsXML schema resolves this design question consistently in favour of elements: the currency is always a child element `<Currency>EUR</Currency>`, not an attribute.

2.3.3 Text, mixed content, CDATA, and comments

Element content falls into three categories. An element may contain only other elements (*element-only content*, the usual case for structured data such as a portfolio), only text (*text-only content*, typical for leaf fields such as `<ISIN>`), or a mixture (*mixed content*, in which text and child elements are interleaved). Mixed content is common in fact-sheet prose and regulatory narrative text, but rare in portfolio and transaction structures.

A short fact-sheet paragraph might look like this:

```
<InvestmentPolicy xml:lang="en">
  The fund invests primarily in large-capitalisation European equities
  and uses a <Benchmark>MSCI Europe NR</Benchmark> as its reference.
</InvestmentPolicy>
```

When text must contain characters that XML would otherwise interpret — for example, a literal `<` or `&` — two escape mechanisms are available. The first is the entity-reference mechanism already mentioned (`<`, `&`). The second is a `CDATA` section, which tells the parser "everything inside here is raw text, don't try to interpret it":

```
<LegalText><![CDATA[
  The management fee is 1.25% per annum of the NAV
  & is accrued daily.
]]></LegalText>
```

`CDATA` is most useful when embedding pre-formatted text, such as an HTML fragment or a legal notice, that contains many XML-special characters.

Comments are written `<!-- like this -->` and are ignored by parsers. They are invaluable in schemas and in hand-edited sample files, but should not be relied upon to carry information that downstream consumers need, because many XML tool chains discard them silently. Processing instructions of the form `<?target data?>` also exist; FundsXML does not use them beyond the XML declaration itself, and a practitioner need only recognise them.

2.4 Character Encoding and the XML Declaration

The first line of almost every XML file is the *XML declaration*:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Three things are worth knowing about it.

First, the declaration is strictly optional under XML 1.0 when the encoding is UTF-8 or UTF-16, but in practice every serious tool chain emits it. Always include it.

Second, the `encoding` attribute tells the parser how the bytes in the file map onto characters. UTF-8 is the default in practice and should be the default in your own production of FundsXML files. The alternatives — ISO-8859-1, Windows-1252, UTF-16 — are legacy choices that cause avoidable pain, most commonly when a German umlaut, a French accent, or a non-breaking space survives a round trip through a tool chain that guesses the encoding wrongly.

Third, UTF-8 files may be preceded by a three-byte *byte-order mark* (the BOM, `EF BB BF`). Some Windows tools write the BOM by default; some Unix tools refuse to parse a file that begins with it. Neither behaviour is strictly wrong, but the combination is a reliable source of production incidents. The safe rule for FundsXML work is: **emit UTF-8 without a BOM**, and treat the appearance of a BOM in an incoming file as a yellow flag worth investigating.

Encoding is not an academic concern for the fund industry. Multi-lingual fund names, legal disclaimers with em-dashes and typographic quotes, and regulatory text with accented characters all pass through the same pipeline as the numbers. A single mis-configured character set turns *Société Générale* into *SociÃ©tÃ© GÃ©nÃ©rale*, and a downstream regulator is unlikely to laugh.

2.5 Namespaces

2.5.1 Why namespaces exist

As soon as XML documents began to combine data from multiple sources, a practical problem emerged. Two independently designed vocabularies might both use an element called `Name`, or `Address`, or `Type`. Without some way of telling them apart, a document that mixed them would be ambiguous.

XML namespaces solve this by giving every element and attribute a two-part name: a namespace URI (which acts as a globally unique identifier) and a local name (the short name used in the document). Two elements share a name only if both parts match.

2.5.2 Declaring and using namespaces

A namespace is bound to a prefix (or to the default) by an `xmlns` attribute:

```
<Fund xmlns="http://example.org/illustrative/fund"
      xmlns:sig="http://www.w3.org/2000/09/xmldsig#">
  <Name>Europa Growth Fund</Name>
  <sig:Signature>
    <!-- digital signature content, treated in Chapter 9 -->
  </sig:Signature>
</Fund>
```

Two namespaces are in play here. The default namespace, declared by `xmlns="..."` without a prefix, applies to `<Fund>` and `<Name>`. The second namespace is bound to the prefix `sig:` and applies to `<sig:Signature>`. A consumer that understands both vocabularies can process each subtree independently.

Three points deserve emphasis because they are the source of most namespace-related confusion:

- **The URI is an identifier, not an address.** Nothing lives at `http://example.org/illustrative/fund`. The URI is chosen for uniqueness, not for retrieval. FundsXML's own namespace URIs follow the same convention.
- **Prefixes are local aliases, not part of the name.** The same document could write `xmlns:f="http://example.org/illustrative/fund"` and then use `<f:Fund>` instead of `<Fund>`; both forms identify the same element. Tools that compare XML must compare the expanded names, not the prefixed ones.
- **Default namespaces propagate to descendants.** Once a default namespace is declared on an element, every unprefixed descendant element is in that namespace — until an inner element redeclares it. Attributes are not covered by the default namespace; an unprefixed attribute is in no namespace at all. This asymmetry surprises newcomers and is worth remembering.

2.5.3 Namespace-aware tooling

A recurring field-failure pattern around FundsXML is the use of a tool that is not namespace-aware. Such tools treat `<f:Fund>` and `<Fund>` as different elements purely on the basis of the textual prefix, and produce XPath queries that silently return empty results against files whose authors happen to have chosen a different prefix. Every tool shown in this book — xmllint, oXygen, IntelliJ, VS Code, and the FreeXmlToolkit treated in Chapter 11 — is namespace-aware. If you ever find yourself writing an XPath expression against a FundsXML document that matches nothing despite an obvious element being present, the first diagnosis should always be: *am I missing a namespace binding?*

2.6 From Well-Formed to Valid: Enter XSD

Well-formedness asks whether a document obeys XML's syntactic rules. *Validity* asks a stronger question: does the document obey the rules of some particular vocabulary? Those rules — which elements may appear, in what order, how often, with what types of content, and with what attributes — are written in a *schema*, and the act of checking a document against that schema is *schema validation*.

XML has more than one schema language. Three are worth knowing by name:

- **DTD (Document Type Definition)** is the original schema language, inherited from SGML. It is expressive but crude: it has no notion of data types beyond "string", no namespace support worth using, and a syntax that is not itself XML. DTDs survive in legacy corners of publishing and HTML but are a historical curiosity for FundsXML purposes.
- **Relax NG** is an elegant, pattern-based schema language that is technically superior to XSD in several ways and is used, for example, by OpenDocument. Its weaker tooling support in enterprise environments has limited its adoption in financial standards.
- **XSD (XML Schema Definition)**, governed by the W3C, is the schema language FundsXML uses. It is itself an XML vocabulary — meaning schemas can be parsed by the same tools that parse instance documents — and it supports a rich type system, namespaces, inheritance, and modularisation.

FundsXML chose XSD for pragmatic reasons. XSD is supported by every serious XML tool, every major programming language, and every enterprise integration platform. Its type system is expressive enough to enforce formats such as ISIN, date, and decimal precision, and its modularisation features scale to the dozens of schema files that make up a modern FundsXML release. No other schema language would have been as durable a choice in 2001, and none has meaningfully displaced it in the two decades since.

The rest of this chapter therefore teaches XSD, and only XSD. When the book speaks of "the schema" from Chapter 3 onwards, it means an XSD schema.

2.7 XSD Simple Types

XSD divides types into two categories. A *simple type* is a type that constrains a single leaf value — the text content of an element or the value of an attribute. A *complex type* describes the structure of an element with child elements and attributes. This section treats simple types; §2.8 treats complex types.

2.7.1 Built-in simple types

XSD provides several dozen built-in simple types. A fund-industry practitioner needs to recognise the following core set on sight:

Table 2.1 — Core XSD built-in types for fund data

Type	Purpose	Example value
<code>xs:string</code>	Arbitrary text	Europa Growth Fund
<code>xs:normalizedString</code>	Text with whitespace normalised	Europa Growth Fund
<code>xs:token</code>	Text with leading, trailing, and multiple internal white-space collapsed	EUR
<code>xs:boolean</code>	true / false (or 1 / 0)	true
<code>xs:decimal</code>	Arbitrary-precision decimal number	123.456789
<code>xs:integer</code>	Whole number	42
<code>xs:double</code>	64-bit floating point	1.2345E3
<code>xs:date</code>	ISO-8601 date	2026-04-30
<code>xs:dateTime</code>	ISO-8601 date with time and optional timezone	2026-04-30T17:30:00+02:00
<code>xs:gYear</code>	A four-digit year	2026
<code>xs:anyURI</code>	A URI (used for namespace references and links)	http://example.org/f
<code>xs:ID</code> / <code>xs:IDREF</code>	A unique identifier / a reference to one	EGF-001

A few observations matter for fund data. `xs:decimal` is the correct type for monetary values and NAVs: it is arbitrary-precision and does not suffer from the rounding surprises of `xs:double`. `xs:date` and `xs:dateTime` require ISO-8601 format — year-month-day, not the assorted national variants; a pipeline that accepts `30/04/2026` is accepting something that is not a valid `xs:date`. `xs:boolean` accepts `true`, `false`, `1`, and `0`, and nothing else; values like `Y` or `Yes` are invalid.

2.7.2 Restricting built-in types

The real power of XSD simple types comes from *restrictions* (formally called *constraining facets*): building a new simple type by constraining an existing one. XSD defines twelve constraining facets, and each facet is applicable only to certain families of types. Table 2.2 gives the complete picture.

Table 2.2 — XSD constraining facets at a glance

Facet	Applies to	Effect
<code>enumeration</code>	all types	Value must be one of a listed set
<code>pattern</code>	all types	Value must match a regular expression
<code>whiteSpace</code>	string types	Controls whitespace handling: <code>preserve</code> , <code>replace</code> , or <code>collapse</code>
<code>length</code>	string, binary, list types	Value must have exactly this length
<code>minLength</code>	string, binary, list types	Value must have at least this length
<code>maxLength</code>	string, binary, list types	Value must have at most this length
<code>minInclusive</code>	numeric, date/time types	Value must be $\geq$ this bound
<code>minExclusive</code>	numeric, date/time types	Value must be $>$ this bound
<code>maxInclusive</code>	numeric, date/time types	Value must be $\leq$ this bound
<code>maxExclusive</code>	numeric, date/time types	Value must be $<$ this bound
<code>totalDigits</code>	<code>xs:decimal</code> and derived	Maximum total number of digits
<code>fractionDigits</code>	<code>xs:decimal</code> and derived	Maximum number of fractional digits

Several facets can be combined in a single restriction — for example, a `pattern` and a `maxLength` on the same type. The XSD processor checks every facet; the value must satisfy all of them.

The following worked examples, drawn from Europa Growth Fund data, illustrate each family of facets.

String restrictions: `pattern` , `length` , `minLength` , `maxLength`

An ISIN pattern. An ISIN is exactly twelve characters: two letters (the country code), followed by nine alphanumeric characters, followed by one decimal digit (the check digit). The XSD that expresses this is:

```
<xs:simpleType name="ISINType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z]{2}[A-Z0-9]{9}[0-9]" />
  </xs:restriction>
</xs:simpleType>
```

A value of `LU0000000001` validates; a value of `LU0000000000` (eleven characters) does not, and neither does `lu0000000001` (lowercase country code). The `pattern` facet uses the regular-expression syntax defined by XSD, which is similar to — but not identical with — the Perl-style regular expressions most programmers know. The most important differences are that XSD patterns always match the *entire* value (no need for `^` and `$` anchors) and that some shorthand character classes differ.

A bounded text field. Many FundsXML text fields carry a maximum-length constraint to prevent absurdly long values from reaching the consumer. A typical text field limited to 256 characters:

```
<xs:simpleType name="Text256Type">
  <xs:restriction base="xs:string">
    <xs:maxLength value="256" />
  </xs:restriction>
</xs:simpleType>
```

For an exact-length field — a two-letter ISO country code, for example — the `length` facet is more appropriate than the combination of `minLength` and `maxLength`:

```
<xs:simpleType name="ISOCountryCodeType">
  <xs:restriction base="xs:string">
    <xs:length value="2" />
    <xs:pattern value="[A-Z]{2}" />
  </xs:restriction>
</xs:simpleType>
```

Here `length` and `pattern` work together: the value must be exactly two characters *and* must consist of two uppercase letters. Either facet alone would be insufficient — `length` alone would accept `12`, and `pattern` alone would accept `ABC` if the expression were written carelessly.

Enumeration: `enumeration`

A currency enumeration. ISO 4217 defines the three-letter alphabetic currency codes. A minimal enumeration covering the currencies in which the Europa Growth Fund issues share classes might be:

```
<xs:simpleType name="CurrencyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="EUR" />
    <xs:enumeration value="USD" />
    <xs:enumeration value="CHF" />
    <xs:enumeration value="GBP" />
  </xs:restriction>
</xs:simpleType>
```



```

    <xs:enumeration value="JPY"/>
  </xs:restriction>
</xs:simpleType>

```

In production, the enumeration would cover the complete ISO 4217 list. The principle is the same: an enumeration is the correct way to pin a field to a closed set of values, and it costs nothing at runtime. Enumerations are not limited to strings — they work on any type. An enumeration of `xs:integer` values, for example, can restrict a field to a fixed set of numeric codes.

Numeric restrictions: `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive`, `totalDigits`, `fractionDigits`

A NAV decimal constraint. A NAV per share must be non-negative, and a practitioner may reasonably insist on no more than eight fractional digits:

```

<xs:simpleType name="NavType">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
    <xs:fractionDigits value="8"/>
  </xs:restriction>
</xs:simpleType>

```

The schema rejects `-1.0` and `123.123456789` (nine fractional digits); it accepts `123.45` and `0`.

The difference between *inclusive* and *exclusive* bounds is exactly what the names suggest. `minInclusive value="0"` accepts zero; `minExclusive value="0"` does not. A percentage field that must be strictly between 0 and 100 would use:

```

<xs:simpleType name="PercentageType">
  <xs:restriction base="xs:decimal">
    <xs:minExclusive value="0"/>
    <xs:maxExclusive value="100"/>
  </xs:restriction>
</xs:simpleType>

```

The `totalDigits` facet limits the total number of significant digits (before *and* after the decimal point), which is useful for monetary amounts where the database column has a fixed precision:

```

<xs:simpleType name="MonetaryAmountType">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="18"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>

```

This type accepts up to sixteen digits before the decimal point and exactly two after — sufficient for any fund-industry amount.

Date and time restrictions: range bounds on `xs:date` and `xs:dateTime`

The same `minInclusive`, `maxInclusive`, `minExclusive`, and `maxExclusive` facets that constrain numbers also constrain dates and date-times. A type for a `ContentDate` that must lie within the twenty-first century:

```
<xs:simpleType name="ReportingDateType">
  <xs:restriction base="xs:date">
    <xs:minInclusive value="2000-01-01"/>
    <xs:maxExclusive value="2100-01-01"/>
  </xs:restriction>
</xs:simpleType>
```

This catches the occasional data-entry error that produces a date in 1026 instead of 2026 — a surprisingly common incident in production.

Whitespace handling: `whiteSpace`

The `whiteSpace` facet controls how the XSD processor normalises whitespace before checking other constraints. Three values are available:

- `preserve` — whitespace is left untouched (the default for `xs:string`).
- `replace` — every tab, carriage return, and newline is replaced with a single space.
- `collapse` — like `replace`, but leading and trailing spaces are also stripped and sequences of multiple spaces are collapsed to one.

In practice, `collapse` is the most useful for fund data, because it prevents invisible whitespace from causing downstream mismatches:

```
<xs:simpleType name="CleanTokenType">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse"/>
    <xs:maxLength value="64"/>
  </xs:restriction>
</xs:simpleType>
```

Note that the built-in type `xs:token` already applies `collapse` semantics, so a `whiteSpace` facet is typically needed only when restricting `xs:string` directly.

Combining facets

Facets compose freely within the limits of applicability. A real-world FundsXML type might combine several:

```
<xs:simpleType name="LEIType">
  <xs:restriction base="xs:string">
    <xs:length value="20"/>
    <xs:pattern value="[A-Z0-9]{18}[0-9]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

A Legal Entity Identifier is exactly twenty characters: eighteen alphanumeric characters followed by two check digits. Both the length and the pattern are enforced; a value that satisfies one but not the other is rejected.

2.7.3 Lists and unions

Two further simple-type constructions are occasionally useful. An `xs:list` type describes a whitespace-separated list of values of another simple type — a natural fit for, say, a list of ISINs on a single line. An `xs:union` type describes a value that may belong to any of several simple types — a natural fit for a field that is either a known enumerated code or a free-form override. Both are used sparingly in FundsXML; you will encounter them rarely and should simply recognise the construction when you do.

2.8 XSD Complex Types and Structuring

A complex type describes an element that has structure: child elements, attributes, or both. It is where the shape of the document is defined. For fund data, almost every element beyond the leaves is a complex type.

2.8.1 The basic form

The workhorse complex type declaration uses `xs:sequence` to say "these children, in this order":

```
<xs:complexType name="FundType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Domicile" type="xs:string"/>
    <xs:element name="BaseCurrency" type="CurrencyType"/>
    <xs:element name="ShareClasses" type="ShareClassesType"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>
```

This type, together with the earlier `CurrencyType` and a `ShareClassesType` we will define in a moment, fully describes the `<Fund>` element from §2.2. The `use="required"` on the attribute says that the `id` attribute must be present.

Note the strong reminder: this `FundType` is an illustrative hand-rolled type for teaching XSD complex-type syntax. The real FundsXML `FundType`, with its `Identifiers`, `Names`, `Currency`, `SingleFundFlag`, `FundStaticData`, `FundDynamicData`, and `SingleFund / Subfunds` choice, is the subject of Chapter 5 and is documented precisely in Appendix C.

2.8.2 Occurrence constraints

Every element declaration can specify how many times it may appear by using `minOccurs` and `maxOccurs`. The defaults are `1` and `1` — exactly once, mandatory. Common variations are:

- `minOccurs="0"` — optional element;
- `minOccurs="0" maxOccurs="unbounded"` — zero or more, the typical shape for a repeated child such as a list of share classes;
- `minOccurs="1" maxOccurs="unbounded"` — one or more;
- explicit numbers — for example, a type that insists on exactly twelve monthly returns would use `minOccurs="12" maxOccurs="12"`.

A `ShareClassesType` that contains one or more `<ShareClass>` children is therefore:

```
<xs:complexType name="ShareClassesType">
  <xs:sequence>
    <xs:element name="ShareClass"
      type="ShareClassType"
      minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

2.8.3 Sequence, choice, and all

`xs:sequence` is the most common compositor, but it is not the only one.

- `xs:sequence` specifies children in a fixed order.
- `xs:choice` specifies that exactly one of several alternatives must appear — used, for example, when a position can be described *either* by its ISIN *or* by a bundle of descriptive fields, but not both.
- `xs:all` specifies that a fixed set of children must each appear exactly once, in any order. It is less often useful than `xs:sequence` and has some restrictions in XSD 1.0 that have limited its adoption.

Compositors can nest: an `xs:sequence` can contain an `xs:choice` that contains another `xs:sequence`, and so on. FundsXML makes extensive use of this nesting to express "either a simple reference to an asset, or a full inline description".

2.8.4 Derivation: extension and restriction

XSD supports two forms of type derivation. *Extension* adds children or attributes to an existing type — the XSD equivalent of subclassing with additional fields. *Restriction* narrows an existing type — the equivalent of tightening a contract.

Extension is the more common of the two in fund data. For example, a hypothetical `EquityPositionType` might extend a generic `PositionType` by adding equity-specific fields:

```
<xs:complexType name="EquityPositionType">
  <xs:complexContent>
    <xs:extension base="PositionType">
      <xs:sequence>
        <xs:element name="Sector" type="xs:string"/>
        <xs:element name="DividendYield" type="xs:decimal" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derivation supports substitution groups and abstract types, which are the mechanism by which FundsXML models "many kinds of position" inside a single portfolio container. Chapter 6 returns to this pattern when it covers the portfolio section in detail.

2.8.5 Global, local, and anonymous

A simple or complex type can be declared *globally*, at the top level of the schema, with a `name` attribute, and then referenced by its name from wherever it is needed. It can also be declared *locally*, anonymously, inside the element declaration it describes. Global types are reusable and make a schema easier to read; anonymous inline types are briefer but cannot be referred to from elsewhere.

FundsXML leans towards global types for anything more than a one-off leaf, and this book follows the same convention in its examples.

2.9 Structuring Larger Schemas

A production schema for a complex domain is never a single file. FundsXML distributes its definitions across several dozen files grouped by concern — control data, funds, portfolios, regulatory modules, and so on. Two mechanisms allow XSD files to refer to one another.

`xs:include` merges another schema file into the current one, under the same namespace. It is the natural choice for splitting a single logical schema into multiple physical files for readability. If `Fund.xsd` defines the fund-level types and `ShareClass.xsd` defines share-class types, and both live in the same namespace, then `Fund.xsd` includes `ShareClass.xsd`:

```
<xs:include schemaLocation="ShareClass.xsd"/>
```

`xs:import` is used when the imported schema lives in a *different* namespace. This is the mechanism by which FundsXML references third-party vocabularies such as XMLDSig for digital signatures or the FinDatEx regulatory templates, each of which has its own namespace. An `import` must mention the foreign namespace explicitly:

```
<xs:import namespace="http://www.w3.org/2000/09/xmlsig#"
  schemaLocation="xmlsig-core-schema.xsd"/>
```

A third, more obscure mechanism — the *chameleon schema* — lets a schema without its own target namespace be included into another and take on the including schema's namespace. It is rare in modern practice and is mentioned here only so that you recognise the name if you encounter it.

Chapter 3 will walk through FundsXML's actual file layout and explain which files import which. For the present chapter, the takeaway is simpler: when you open a schema and see `xs:include` or `xs:import` at the top, you are looking at a composition of several files, and a complete picture of any element's definition may require following the links.

2.10 XML Validation in Practice

Schema validation is the act of checking that an instance document matches a schema. Every mainstream XML tool chain — every IDE plugin, every command-line utility, every library in every programming language — supports it. Performing validation early and often is the single most effective discipline a FundsXML practitioner can adopt.

2.10.1 Two error classes

Validation produces errors of two very different kinds, and learning to distinguish them is half the battle.

Well-formedness errors are syntactic: the file is not even XML. Typical messages are "mismatched tag", "attribute value not quoted", "premature end of data", or "encoding error". A well-formedness error means no further validation is possible — the parser cannot build a tree.

Schema validity errors are structural: the file is valid XML but does not match the schema. Typical messages are "element X not expected here", "attribute Y is required but missing", "value Z does not match pattern", or "element W has invalid child element V".

The distinction matters because the two classes of error are fixed differently. A well-formedness error is almost always the result of a tool that was not designed to emit XML — a template-based generator, a hand edit, a buggy library. A schema validity error is almost always a semantic mismatch: the producer's understanding of the vocabulary diverged from the schema's.

2.10.2 Validating from the command line

The simplest production-grade validator is `xmllint`, part of the `libxml2` library that ships with every modern Linux and macOS system and is readily installed on Windows. A one-line validation against a schema looks like this:

```
xmllint --noout --schema Fund.xsd Fund.xml
```

The `--noout` flag suppresses the default behaviour of reprinting the document on success; without it, a successful validation floods the terminal with the document's content. On success, `xmllint` prints `Fund.xml validates`; on failure, it prints a line-numbered description of each error and exits with a non-zero status.

For continuous-integration pipelines, this is all that is needed. A shell script that loops over the outbound FundsXML files and runs `xmllint` against the schema catches the vast majority of problems at the cheapest possible point — before the files leave the building.

2.10.3 Reading a validation error

Consider a deliberately broken share class, in which the ISIN has been corrupted:

```
<ShareClass>
  <ISIN>LU000000000X</ISIN>
  <Currency>EUR</Currency>
</ShareClass>
```

Validated against the `ISINType` defined in §2.7.2, this produces an error along the following lines:

```
Fund.xml:7: element ISIN: Schemas validity error :
  Element 'ISIN': [facet 'pattern'] The value 'LU000000000X'
  is not accepted by the pattern '[A-Z]{2}[A-Z0-9]{9}[0-9]'.
```

Three pieces of information matter in this message. The file-and-line prefix (`Fund.xml:7`) points at the offending element. The bracketed facet name (`[facet 'pattern']`) says which restriction rejected the value — here, the regular expression pattern. The value itself (`LU000000000X`) is quoted back so you can compare it to your expectation. Together, these three are enough to diagnose most schema-level errors without reading the schema at all.

2.10.4 What schema validation does not catch

Schema validation is powerful but limited. It answers the question *is this document structurally correct?* — not the question *is this document semantically reasonable?* A well-formed, schema-valid FundsXML document may still:

- have a NAV that is positive but implausibly large;
- describe a share class whose currency does not match the base currency of the fund and whose hedged flag is `false`;
- reference an asset by UniqueID that does not appear elsewhere in the document;

- declare a fund-total that does not equal the sum of its position values.

Catching these requires a second layer of checking — *business-rule validation* — which FundsXML implementations typically perform using Schematron or hand-written code. Chapter 10 develops the two-stage validation model in full. For now, it is enough to note that schema validation is necessary but not sufficient, and that no FundsXML production pipeline should ever rely on schema validation alone.

2.11 XPath

Once you have a FundsXML document, you will want to pull specific values out of it. The tool for that job is **XPath**, a compact expression language for navigating the XML tree. XPath is the query language behind almost every XML technology: XSLT, XQuery, Schematron, and the XML facilities of every mainstream programming language. A solid understanding of XPath makes every later chapter of this book easier to follow.

2.11.1 The tree model

XPath sees an XML document as a tree. Every piece of the document is a *node*, and nodes come in seven kinds:

Node type	Example	Notation in XPath
Document (root)	the document itself	<code>/</code>
Element	<code><Fund></code>	<code>Fund</code>
Attribute	<code>ccy="EUR"</code>	<code>@ccy</code>
Text	the characters between tags	<code>text()</code>
Comment	<code><!-- ... --></code>	<code>comment()</code>
Processing instruction	<code><?xml-stylesheet ...?></code>	<code>processing-instruction()</code>
Namespace	<code>xmlns:xsi="..."</code>	<code>namespace::xsi</code>

For fund-data work, element, attribute, and text nodes account for 99 per cent of all queries. An XPath expression starts at a *context node* and walks the tree from there, producing a *node set* — a collection of zero or more nodes.

2.11.2 Navigation: paths, steps, and shortcuts

Absolute paths start with a single `/` at the document root and walk downward by element name:

- `/FundsXML4` — the root element.
- `/FundsXML4/ControlData/ContentDate` — the content date of the delivery.

- `/FundsXML4/Funds/Fund` — every `Fund` element under the root.

Relative paths start from the current context node. In most command-line tools, the context node is the document root, so the effect is the same as an absolute path; inside XSLT templates and predicates, the context node is whatever node the template is currently processing.

The double-slash `//` is the most-used shortcut in practice. It means "anywhere below", at any depth:

- `//ShareClass` — every `ShareClass` element, regardless of where it sits in the tree.
- `//ISIN` — every `ISIN`, however deeply nested.

The dot `.` refers to the current context node; **the double dot** `..` refers to the parent node. Both are used constantly inside predicates and XSLT:

- `.` — the current node.
- `..` — the parent of the current node.
- `../Currency` — the `Currency` sibling of the current node's parent.

Attribute access uses the `@` prefix:

- `//Amount/@ccy` — the `ccy` attribute of every `Amount` element.
- `//@ccy` — every `ccy` attribute anywhere in the document.

Wildcards select nodes regardless of name:

- `//Fund/*` — every direct child element of every `Fund`, whatever the element name.
- `//@*` — every attribute in the document.
- `//Fund/node()` — every child node (elements, text, comments) of every `Fund`.

Text access uses `text()`:

- `//ISIN/text()` — the string value of every `ISIN` element.

2.11.3 Predicates and filtering

Predicates, in square brackets, filter node sets by arbitrary conditions. They are the `WHERE` clause of XPath.

Value comparisons:

- `//ShareClass[Currency='EUR']` — share classes whose `Currency` child has text content `EUR`.
- `//Fund[Identifiers/LEI='549300ABCDEFGHJI1234']` — the fund whose `LEI` matches.
- `//Position[MarketValue > 1000000]` — positions with a market value above one million.

Positional predicates:

- `//ShareClass[1]` — the first share class (XPath positions start at 1, not 0).
- `//ShareClass[last()]` — the last.
- `//ShareClass[position() <= 3]` — the first three.

Existence tests — a predicate that names a path is true if the path selects at least one node:

- `//ShareClass[Identifiers/ISIN]` — share classes that have an ISIN.
- `//Fund[not(FundDynamicData)]` — funds that have no dynamic data.

Compound predicates combine conditions with `and`, `or`, and `not()`:

- `//Position[MarketValue > 1000000 and Currency='EUR']` — EUR positions above one million.
- `//ShareClass[Currency='EUR' or Currency='CHF']` — EUR or CHF share classes.

Chained predicates apply successive filters:

- `//ShareClass[Currency='EUR'][1]` — the first EUR share class.

2.11.4 Axes: navigating in every direction

Each step in an XPath expression uses an *axis* that determines the direction of travel through the tree. Most of the time the axis is implicit — `Fund/Currency` is shorthand for `Fund/child::Currency` — but the full axis syntax is needed when you must navigate sideways or upward.

Table 2.4 — XPath axes

Axis	Direction	Shorthand	Example
<code>child::</code>	direct children	<i>(default)</i>	<code>child::Fund = Fund</code>
<code>attribute::</code>	attributes	<code>@</code>	<code>attribute::ccy = @ccy</code>
<code>parent::</code>	parent node	<code>..</code>	<code>parent::Fund</code>
<code>self::</code>	current node	<code>.</code>	<code>self::Fund</code>
<code>descendant::</code>	all descendants	<code>—</code>	<code>descendant::ISIN</code>
<code>descendant-or-self::</code>	self + descendants	<code>//</code>	<code>descendant-or-self::node()</code>
<code>ancestor::</code>	all ancestors	<code>—</code>	<code>ancestor::Fund</code>
<code>ancestor-or-self::</code>	self + ancestors	<code>—</code>	<code>ancestor-or-self::Funds</code>
<code>following-sibling::</code>	later siblings	<code>—</code>	<code>following-sibling::ShareClass</code>
<code>preceding-sibling::</code>	earlier siblings	<code>—</code>	<code>preceding-sibling::ShareClass</code>
<code>following::</code>	everything after	<code>—</code>	<code>following::Fund</code>
<code>preceding::</code>	everything before	<code>—</code>	<code>preceding::Fund</code>

The axes that matter most for fund-data work are `child` (the default), `attribute (@)`, `parent (..)`, `descendant-or-self (//)`, and `following-sibling / preceding-sibling`. A practical example: given a `Position` element, find the fund it belongs to:

- `ancestor::Fund/Names/OfficialName` — walks up the tree to the enclosing `Fund` and then back down to its name.

2.11.5 Functions

XPath 1.0 provides a compact function library. The functions most relevant to fund-data work fall into four groups.

String functions:

Function	Purpose	Example
<code>concat(a, b, ...)</code>	Concatenate strings	<code>concat(//LEI, ' - ', //OfficialName)</code>
<code>contains(s, sub)</code>	Test if <code>s</code> contains <code>sub</code>	<code>//Fund[contains(Names/OfficialName, 'Growth')]</code>
<code>starts-with(s, pre)</code>	Test if <code>s</code> starts with <code>pre</code>	<code>//ISIN[starts-with(., 'LU')]</code>
<code>substring(s, pos, len)</code>	Extract substring	<code>substring(//ISIN, 1, 2) → country code</code>
<code>string-length(s)</code>	Length of string	<code>//Name[string-length(.) > 100]</code>
<code>normalize-space(s)</code>	Strip and collapse whitespace	<code>normalize-space(//OfficialName)</code>
<code>translate(s, from, to)</code>	Character-by-character replacement	<code>translate(//Currency, 'abcdefghijklmnopqrstu-vwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')</code>

Numeric functions:

Function	Purpose	Example
<code>sum(node-set)</code>	Sum of numeric values	<code>sum(//Position/MarketValue)</code>
<code>count(node-set)</code>	Number of nodes	<code>count(//ShareClass)</code>
<code>round(n)</code>	Round to nearest integer	<code>round(//NavPerShare)</code>
<code>floor(n)</code>	Round down	<code>floor(//NavPerShare)</code>
<code>ceiling(n)</code>	Round up	<code>ceiling(//NavPerShare)</code>
<code>number(s)</code>	Convert to number	<code>number(//TotalNetAssetValue/Amount)</code>

Boolean functions:

Function	Purpose	Example
<code>not(expr)</code>	Logical negation	<code>//Fund[not(FundDynamicData)]</code>
<code>true() / false()</code>	Boolean constants	predicate building blocks
<code>boolean(expr)</code>	Convert to boolean	<code>boolean(//ISIN) → true if any ISIN exists</code>

Node functions:

Function	Purpose	Example
<code>name()</code>	Element name (with prefix)	<code>//Fund/*[name()='Currency']</code>
<code>local-name()</code>	Element name (without prefix)	useful in namespaced documents
<code>namespace-uri()</code>	Namespace URI of the node	debugging namespace issues

Illustrated against the Europa Growth Fund document from §2.2: `count(//ShareClass)` returns 2, `sum(//ShareClass/NavPerShare)` returns the sum of per-share NAVs, and `concat(//OfficialName, '(', //Currency, ')')` returns Europa Growth Fund (EUR).

2.11.6 Operators and expressions

XPath expressions support arithmetic, comparison, boolean logic, and the union operator.

Arithmetic — `+`, `-`, `*`, `div`, `mod`:

- `//MarketValue div //TotalNetAssetValue/Amount * 100` — position weight as a percentage.
- `count(//Position) mod 2` — is the number of positions odd or even?

Note that division uses `div`, not `/`, because `/` is already taken as the path separator.

Comparison — `=`, `!=`, `<`, `>`, `<=`, `>=` :

- `//Position[Quantity > 10000]` — positions with more than ten thousand units.
- `//TotalAssetValue[NavDate = '2026-03-31']` — the asset value for a specific date.

When used inside XML attributes (as in XSLT), the `<` character must be escaped as `<` .

Boolean logic — `and`, `or`, `not()` :

- `//Position[Quantity > 10000 and MarketValue > 5000000]` — large positions by both measures.

Union — `|` combines two node sets:

- `//Fund/Names/OfficialName | //Fund/Names/ShortName` — all official and short names together.

2.11.7 XPath and namespaces

XPath 1.0 matches elements by their expanded name (namespace URI + local name), not by their textual prefix. When a document uses a namespace — as many real XML documents do — your XPath tooling must be told which prefix in the query stands for which namespace URI. Every XPath-capable tool has its own way of declaring the binding; the mechanism differs, but the obligation is the same. An XPath query written without a namespace binding against a namespaced document is the single most common source of "no results, but I can see the element right there" frustration.

2.11.8 XPath versions

XPath has evolved through several versions, and the differences matter for tool selection.

XPath 1.0 (1999) is the version described in this section. It is universally supported by every XML library, every IDE, and every command-line tool. It works with node sets, strings, numbers, and booleans. For the purposes of reading and querying FundsXML, it is sufficient.

XPath 2.0 (2007) adds a richer type system (sequences instead of node sets, typed values aligned with XSD), `if/then/else` expressions, `for` expressions, quantified expressions (`some/every`), and a much larger function library including regular-expression matching (`matches()` , `replace()` , `tokenize()`). It is the version used by XSLT 2.0, XQuery, and Schematron.

XPath 3.0 / 3.1 (2014/2017) adds higher-order functions (functions as values), the `let` expression, arrow operator (`=>`), inline functions, and maps and arrays. It is used by XSLT 3.0 and XQuery 3.1.

For day-to-day FundsXML work, XPath 1.0 covers the vast majority of queries. XPath 2.0 becomes useful when writing Schematron business rules (Chapter 10) or complex XSLT transformations (§2.12), where its richer type system and `matches()` function simplify the work considerably.

2.12 XSLT Transformations

XSLT — the Extensible Stylesheet Language Transformations — is the standard way of turning one XML document into another document, which may itself be XML, HTML, CSV, plain text, or almost anything else. In a FundsXML context, XSLT is how a single canonical fund-data document becomes a fact-sheet HTML fragment, a distributor-specific CSV, a regulatory submission, or a formatted printout.

2.12.1 The mental model

An XSLT stylesheet is itself an XML document. It contains a set of *templates*, each of which says "when you encounter a node that matches *this pattern*, produce *this output*". The processor walks the input tree, matches each node against the templates in turn, and writes the output. The model is declarative, not imperative: you describe the mapping, and the processor decides how to traverse the source.

The three templates you will see most often are `xsl:template`, which defines a match rule; `xsl:apply-templates`, which tells the processor to recursively process child nodes; and `xsl:value-of`, which inserts the string value of a node into the output.

2.12.2 Example: FundsXML to CSV

Here is a minimal stylesheet that turns the Europa Growth Fund document from §2.2 into a two-column CSV of ISINs and currencies:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="UTF-8"/>

  <xsl:template match="/FundsXML4">
    <xsl:text>ISIN;Currency&#10;</xsl:text>
    <xsl:apply-templates select="Funds/Fund/SingleFund/ShareClasses/ShareClass"/>
  </xsl:template>

  <xsl:template match="ShareClass">
    <xsl:value-of select="Identifiers/ISIN"/>
    <xsl:text>;</xsl:text>
    <xsl:value-of select="Currency"/>
    <xsl:text>&#10;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

The `xsl:output method="text"` declaration switches the processor into plain-text mode, so that the output is written directly without XML escaping. The root template matches the `FundsXML4` root element, writes a header line, and then delegates to the share-class template for each `ShareClass` child. The share-class template writes one CSV line per input share class. Against the §2.2 document the result is:

```

ISIN;Currency
LU1234567890;EUR
LU1234567892;CHF

```

A real counterparty feed, of course, requires more columns. The following stylesheet produces a richer export that includes the fund name, the share-class name, the LEI of the fund, and the NAV date — all the fields a distributor typically needs in a daily share-class listing:

```

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="UTF-8"/>

  <xsl:template match="/FundsXML4">
    <xsl:text>FundName;LEI;ISIN;ShareClassName;Currency;NavDate&#10;</xsl:text>
    <xsl:apply-templates select="Funds/Fund"/>
  </xsl:template>

  <xsl:template match="Fund">
    <xsl:variable name="fundName" select="Names/OfficialName"/>
    <xsl:variable name="lei" select="Identifiers/LEI"/>
    <xsl:variable name="navDate"
      select="FundDynamicData/TotalAssetValues/TotalAssetValue/NavDate"/>
    <xsl:for-each select="SingleFund/ShareClasses/ShareClass">
      <xsl:value-of select="$fundName"/>
      <xsl:text>;</xsl:text>
      <xsl:value-of select="$lei"/>
      <xsl:text>;</xsl:text>
      <xsl:value-of select="Identifiers/ISIN"/>
      <xsl:text>;</xsl:text>
      <xsl:value-of select="Names/OfficialName"/>
      <xsl:text>;</xsl:text>
      <xsl:value-of select="Currency"/>
      <xsl:text>;</xsl:text>
      <xsl:value-of select="$navDate"/>
      <xsl:text>&#10;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>

```

Two new XSLT features appear. First, `xsl:variable` captures values from the enclosing `Fund` node so that they can be reused inside the inner `xsl:for-each` loop without repeating the full XPath each time. Second, `xsl:for-each` iterates over the share classes directly rather than delegating to a separate template — a stylistic choice that works well when the logic is short and self-contained. Against the §2.2 document the output is:

```

FundName;LEI;ISIN;ShareClassName;Currency;NavDate
Europa Growth Fund;549300ABCDEFGHIJ1234;LU1234567890;Europa Growth Fund I EUR
Acc;EUR;2026-03-31

```

```
Europa Growth Fund;549300ABCDEFGHIJ1234;LU1234567892;Europa Growth Fund R CHF Acc
Hedged;CHF;2026-03-31
```

This is the beating heart of the distribution pipeline. Every bilateral CSV feed in an unstandardised production, of the kind Chapter 1 catalogued, can be replaced by a single canonical FundsXML file plus one XSLT per counterparty. If Distributor A wants semicolons and Distributor B wants tabs, the difference is a single character in one `xsl:text` element. Changes to any one counterparty touch one stylesheet and nothing else.

2.12.3 Example: FundsXML to fact-sheet HTML

A second stylesheet produces a fact-sheet page from the same Europa Growth Fund document. Unlike the CSV, this time the output is a complete HTML document with a title, a summary table of fund-level metadata, and a share-class listing:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" encoding="UTF-8"/>

  <xsl:template match="/">
    <xsl:variable name="fund" select="/FundsXML4/Funds/Fund"/>
    <xsl:variable name="nav"
      select="$fund/FundDynamicData/TotalAssetValues/TotalAssetValue
        /TotalNetAssetValue/Amount"/>
    <html>
      <head><title><xsl:value-of select="$fund/Names/OfficialName"/></title></head>
      <body>
        <h1><xsl:value-of select="$fund/Names/OfficialName"/></h1>
        <table border="1" cellpadding="4">
          <tr><td>LEI</td>
            <td><xsl:value-of select="$fund/Identifiers/LEI"/></td></tr>
          <tr><td>Currency</td>
            <td><xsl:value-of select="$fund/Currency"/></td></tr>
          <tr><td>NAV</td>
            <td><xsl:value-of select="$nav"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="$nav/@ccy"/></td></tr>
          <tr><td>NAV Date</td>
            <td><xsl:value-of select="$fund/FundDynamicData
              /TotalAssetValues/TotalAssetValue/NavDate"/></td></tr>
          <tr><td>Content Date</td>
            <td><xsl:value-of
              select="/FundsXML4/ControlData/ContentDate"/></td></tr>
        </table>

        <h2>Share classes</h2>
        <table border="1" cellpadding="4">
          <tr><th>ISIN</th><th>Name</th><th>Currency</th></tr>
          <xsl:for-each select="$fund/SingleFund/ShareClasses/ShareClass">
            <tr>
```



```

        <td><xsl:value-of select="Identifiers/ISIN"/></td>
        <td><xsl:value-of select="Names/OfficialName"/></td>
        <td><xsl:value-of select="Currency"/></td>
    </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

Several things are worth noting. The root template matches `/` rather than `/FundsXML4`; either form works, but matching the document root is slightly more idiomatic when the stylesheet produces a complete HTML page. The `$fund` and `$nav` variables keep the XPath expressions short and readable inside the template body. The `$nav/@ccy` expression reads the `ccy` attribute from the `Amount` element, so the currency appears next to the value without being hard-coded.

The `method="html"` output declaration switches the processor into HTML-serialisation mode, which is subtly different from XML: empty elements like `<br>` are written without the self-closing slash, boolean attributes are written bare, and character references are emitted in a form that browsers expect. Against the §2.2 document, the result is a self-contained HTML page whose summary table reads:

LEI	549300ABCDEFGHJIJ1234
Currency	EUR
NAV	464552848.78 EUR
NAV Date	2026-03-31
Content Date	2026-03-31

and whose share-class table lists both share classes with their ISINs, names, and currencies. A production fact sheet would add styling, logos, and far more data — but the principle is the same: the XSLT carries the layout logic, and the FundsXML document supplies the data.

2.12.4 Example: data-quality checks with XSLT

The examples so far transform FundsXML into a different *format* — CSV, HTML. But XSLT can just as well transform a document into a *report about itself*. This opens the door to a lightweight data-quality layer: encode your business rules as an XSLT stylesheet, run it against every incoming file, and inspect the output for violations.

Consider one of the most fundamental fund-data invariants: *the sum of the market values of all portfolio positions must equal the fund's total net asset value on the same date*. If a FundsXML file carries both a `TotalNetAssetValue` and a set of `Position` elements with `TotalValue` amounts, the following stylesheet checks whether they agree:

```

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="UTF-8"/>

  <xsl:template match="/FundsXML4">
    <xsl:for-each select="Funds/Fund">
      <xsl:variable name="fundName" select="Names/OfficialName"/>
      <xsl:variable name="fundCcy" select="Currency"/>
      <xsl:variable name="fundNAV"
        select="FundDynamicData/TotalAssetValues/TotalAssetValue
          /TotalNetAssetValue/Amount[@ccy = $fundCcy]"/>
      <xsl:variable name="positionSum"
        select="sum(FundDynamicData/Portfolios/Portfolio
          /Positions/Position/TotalValue/Amount[@ccy = $fundCcy])"/>
      <xsl:variable name="diff" select="$fundNAV - $positionSum"/>

      <xsl:value-of select="$fundName"/>
      <xsl:text>&#10; Fund NAV: </xsl:text>
      <xsl:value-of select="$fundNAV"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="$fundCcy"/>
      <xsl:text>&#10; Position sum: </xsl:text>
      <xsl:value-of select="$positionSum"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="$fundCcy"/>
      <xsl:text>&#10; Difference: </xsl:text>
      <xsl:value-of select="$diff"/>
      <xsl:choose>
        <xsl:when test="$diff = 0">
          <xsl:text>&#10; Result: PASS&#10;</xsl:text>
        </xsl:when>
        <xsl:when test="$diff > -1 and $diff < 1">
          <xsl:text>&#10; Result: PASS (rounding)&#10;</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>&#10; Result: FAIL&#10;</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>

```

The stylesheet iterates over every **Fund** in the document, computes the sum of position values in the fund's own currency, and compares it with the reported total NAV. A difference below one currency unit is accepted as a rounding tolerance; anything larger is flagged as **FAIL**. The output is plain text, easily piped into a log file or a monitoring system:

```

Europa Growth Fund
Fund NAV:      464552848.78 EUR

```

```
Position sum: 464552848.78 EUR
Difference:    0
Result:       PASS
```

The same principle scales to any business rule that can be expressed as a combination of XPath queries: *does every share class carry an ISIN?, is the NAV date consistent with the content date?, do percentage allocations sum to 100?* Each rule becomes one additional block in the stylesheet. The output can be plain text, as above, or HTML for a visual dashboard — or even XML, if the quality report is to be consumed by another automated step.

This approach has a practical advantage: it requires nothing beyond an XSLT 1.0 processor, which is already available on every platform (§2.12.6). There is no need to learn a separate validation language. For organisations that prefer a more formal, standards-based approach to business-rule validation, Schematron — which Chapter 10 covers in detail — provides a dedicated assertion language on top of XPath and can be compiled to XSLT automatically. But for teams that already know XSLT, writing quality checks directly is a natural and effective starting point.

The FundsXML working group maintains a public repository of ready-to-use XSLT stylesheets for data-quality checking at github.com/fundxml/examples. It includes both a basic five-section check (XSLT 2.0) and a comprehensive ten-section HTML dashboard (XSLT 1.0) that covers NAV reconciliation, share-class price verification, percentage-allocation checks, asset-specific validations, and more. These stylesheets can be used as-is against any FundsXML 4.x document or adapted to an organisation's specific rules.

2.12.5 The identity transform

One small but powerful idiom deserves mention. The *identity transform* is a stylesheet that simply copies its input to its output unchanged:

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>
```

Combined with one or two additional templates that override specific nodes, the identity transform is the standard way of making surgical changes to a FundsXML document — for example, renaming a single element or stripping a specific attribute — without touching anything else. Any XSLT tutorial that goes beyond a few pages will show it again and again.

2.12.6 Versions

XSLT exists in three main versions. XSLT 1.0 is universally supported and is enough for almost every practical FundsXML transformation a fund practitioner will write. XSLT 2.0 adds sequences, richer types, and grouping constructs. XSLT 3.0 adds streaming (important for very large documents) and higher-order functions. The code in this book uses XSLT 1.0 unless explicitly noted; if you have access to a 2.0 or 3.0 processor, you lose nothing by running 1.0 stylesheets on it.

2.13 Tools for Day-to-Day XML Work

A practitioner need not build tools from scratch; a rich ecosystem already exists. This section gives the lie of the land; Chapter 11 revisits the tools in depth, with installation guides, configuration advice, and feature-by-feature comparisons.

Table 2.2 — XML tooling at a glance

Tool	Kind	Strengths	Treated in depth
xmllint	Command-line	Validation, formatting, XPath; ships with libxml2; free	§2.10, Chapter 10
oXygen XML Editor	Commercial IDE	Visual schema design, schema-aware editing, XSLT debugging	Chapter 11
Altova XMLSpy	Commercial IDE	Visual design, generation, data mapping	Chapter 11
IntelliJ IDEA / Ultimate	General IDE	Strong XML and XSD support, integrated validation	Chapter 11
VS Code (+ XML plugins)	General editor	Free; lightweight; good validation with the Red Hat XML plugin	Chapter 11
Eclipse (+ Web Tools)	General IDE	Schema view, XML editors, free	Chapter 11
FreeXmlToolkit	FundsXML-aware	Purpose-built for FundsXML workflows	Chapter 11

Two points about this table matter now. First, all of these tools are namespace-aware and all support XSD validation against the FundsXML schemas. The choice between them is, to a first approximation, a choice of ergonomics and cost, not of capability. Second, the free options — `xmllint`, VS Code with the Red Hat plugin, and the FreeXmlToolkit — are enough to reach a fully operational FundsXML pipeline. The commercial IDEs become attractive when schema *authoring* and visual *data mapping* become part of the daily workflow, not merely schema *consumption*.

Beyond this chapter. A knowledgeable reader will notice that this chapter has stopped short of several valid XML topics: XML Catalogs, XInclude, XPointer, XQuery, Schematron, the XSD 1.1 assertion facility, SOAP and WSDL, and XSLT 2.0/3.0 features. None of them are needed to read, validate, or transform FundsXML, and each has been omitted deliberately to keep the chapter short. Where a later chapter reaches for one of them — Schematron in Chapter 10, XMLDSig in Chapter 9 — we will introduce it at that point, in the context where it matters.

2.14 Common Pitfalls

Every experienced FundsXML practitioner keeps a mental list of recurring traps. The list below is not exhaustive, but it covers the failures that appear most frequently in production.

- **Encoding mismatches.** A file declared as UTF-8 but actually written as Windows-1252 will parse — until the first accented character. Always emit UTF-8, always read it with a UTF-8-aware parser, and never paste text between editors that disagree on their default encoding.
- **The byte-order mark.** A BOM at the start of a UTF-8 file is legal but is rejected by some downstream tools and silently corrupts some others. Configure your editor to save UTF-8 without a BOM.
- **Namespace-unaware XPath.** If an XPath query returns an empty result set against a document in which the element is clearly present, the first check is whether the namespace has been declared to the XPath processor. It almost always has not.
- **Attribute vs element confusion.** Two producers of the same logical data choose differently — one writes `<Currency>EUR</Currency>`, the other `<Amount currency="EUR">100</Amount>` — and a consumer that expects one form silently ignores the other. The schema is authoritative; read it.
- **Decimal-separator locale.** A decimal number written as `1.234,56` is not an `xs:decimal`. European locales that use the comma for the decimal separator and the period for the thousands separator must convert before emitting XML.
- **Whitespace and the silent trim.** `xs:token` collapses internal whitespace; `xs:string` preserves it. An ISIN that someone has appended a stray trailing space to will fail a pattern validation but can pass an `xs:token` comparison — or vice versa, depending on the order.
- **Case-sensitivity of enumerations.** `EUR`, `eur`, and `Eur` are three different values to an XSD enumeration. Converting incoming free text to a normalised form before validating is a defensive habit worth acquiring.
- **Comments as data.** Comments are stripped by many processors. Do not put anything in a comment that you expect to be available downstream.
- **Large files and memory.** A DOM-based parser loads the entire document into memory. For the hundreds-of-megabytes files that a fund-of-funds look-through can produce, a streaming parser (SAX, StAX, or XSLT 3.0 streaming) is the right tool.

Each of these pitfalls has cost someone a production incident at some point. Recognising the pattern in your own pipelines is half the defence.

2.15 Key Takeaways

- XML represents data as a tree of named elements that may carry attributes and contain text; the whole technical apparatus of this chapter is layered on top of this simple model.
- A document is *well-formed* if it obeys XML's syntactic rules, and *valid* if it additionally matches a schema. Well-formedness is a precondition for validation; validation is the first line of defence against production errors.

- XML namespaces give elements and attributes globally unique identities, and XSD is the schema language in which FundsXML is written; together they enable the composition of independently maintained vocabularies into a single document.
- XSD simple types — with restrictions such as `pattern`, `enumeration`, `minInclusive`, and `fractionDigits` — are the correct mechanism for enforcing formats like ISIN, currency, and NAV precision.
- XSD complex types, built from `xs:sequence`, `xs:choice`, `xs:all`, and element occurrences, describe the shape of every non-leaf element. Type derivation lets specialised types extend or restrict general ones, and `xs:include` and `xs:import` stitch large schemas together from many files.
- Schema validation is powerful but partial; semantic rules that XSD cannot express belong to a second, business-rule validation layer that Chapter 10 develops in full.
- XPath in its 1.0 dialect is enough for almost every fund-data query a practitioner will need to write, and XSLT 1.0 is enough for almost every transformation.
- A small, well-chosen toolchain — a namespace-aware editor, a command-line validator such as `xmllint`, and a FundsXML-aware tool such as the FreeXmlToolkit — is all the equipment required to work productively with FundsXML. Chapter 11 revisits each tool in depth.

With the XML and XSD vocabulary in place, we are ready to look at FundsXML itself — its history, its organisation, its architecture, and its schema layout — which is the subject of Chapter 3.



PART I — FOUNDATIONS · CHAPTER 3

FundsXML — The Standard at a Glance

History, organisation, and architecture

3.1 Setting the Scene: From Syntax to Substance

Chapter 1 made the case for a shared standard. Chapter 2 introduced XML and XSD as the technologies on which such a standard can be built. We now bring those two threads together and ask the obvious next question: *what exactly is FundsXML?*

Return once more to the Europa Growth Fund. After the month-end episode that opened this book, and after a thirty-page crash course in XML, the operations team gathers around a whiteboard for what the head of operations calls a "ten-minute orientation". The team has, by this point, read a number of FundsXML deliveries with a text editor and even run one or two against a schema validator. They can see that the files are well-formed and, mostly, that they are valid. What they still cannot see is the *shape of the thing as a whole*. Where did this schema come from? Who decides what belongs in it and what does not? How are the pieces arranged, and why in that particular way? Unless these questions are answered first, every later chapter of this book will feel like a tour through rooms whose floor plan the reader has never seen.

This chapter is that floor plan. It closes Part I of the book with a deliberately high-altitude view of FundsXML. We tell the story of how the standard came into being and who maintains it today; we distil the design principles that shape every FundsXML message; and we draw a mental map of the five main areas of the schema. No field-level detail appears here — that begins in Chapter 4. The aim is orientation, not completeness: by the end of the chapter the reader should be able to look at any FundsXML document, identify the five main areas, and know which later chapter of this book covers each of them.

By the end of this chapter, you should be able to:

- tell the story of FundsXML from its German origins in 2001 through to the current 4.2.x release of 2026;
- name the bodies and working groups that maintain the standard and explain how a change gets into the schema;
- state the five core design principles that shape every FundsXML delivery;
- draw, from memory, a mental map of the five main schema areas — *ControlData*, *Funds*, *AssetMasterData*, *Documents*, and *RegulatoryReportings* — and recognise how they fit together;
- read a short FundsXML document skeleton on sight and point at the area each part belongs to.

The chapter is built around three questions — *where does FundsXML come from?*, *who runs it?*, and *what does it look like?* — in that order. Readers in a hurry may skim §3.2 and §3.3 and concentrate on §3.4 to §3.6, which are the structural core of the chapter and the foundation for Part II.

3.2 A Brief History of FundsXML

Standards never arrive out of nowhere. They are always the answer to a concrete industrial problem, and the character of the answer is shaped by the people who formulated the question. FundsXML is no exception. Its story falls naturally into three waves: a German origin phase, a European expansion that crossed national borders, and a regulatory era driven by the wave of EU fund-related legislation of the last decade.

3.2.1 Origins: Germany, 2001–2005

FundsXML was created in 2001 by a group of major German-speaking fund-management houses seeking a common format for the fund-data they all had to exchange with one another. The founding participants were **Allianz Global Investment**, **COMINVEST** (later absorbed into Allianz Global Investment), **Credit Suisse Asset Management**, **DEKA**, **DWS**, and **Union Investment**, with early technical support from **Vontobel**, the Swiss software vendor **K&W Software AG**, and **FERI TRUST**. The immediate trigger was neither regulatory nor ideological; it was operational. Each of these houses managed thousands of fund records, and each bilateral channel to a distributor, a depository, a data vendor, or the financial press used a different CSV layout, a different Excel template, or a different proprietary format. Every new distribution contract added a new mapping, and every mapping had to be maintained forever.

The initial response was pragmatic. A small working group drafted a lightweight XML vocabulary that described the master data and daily price data of a fund, agreed on it informally, and began to use it in production. There was no standards body at first, no formal certification, and no public repository — only a shared schema file, a shared understanding of what the elements meant, and a willingness to keep the format stable for long enough that the investment in implementation would pay off. This informal start left two enduring marks on the standard's character: a preference for practical problem-solving over theoretical elegance, and an instinct for backwards compatibility that has survived every subsequent revision.

The informal phase did not last long. By late 2003, the **BVI** (*Bundesverband Investment und Asset Management e.V.*), the German fund-industry association, had recognised that its members needed a single supported standard for fund-data exchange and formally adopted the effort. **Version 1.0 of FundsXML was published on 8 December 2003** under BVI stewardship. That publication is the moment at which FundsXML ceased to be a private inter-company schema and became a named, versioned industry standard.

3.2.2 European Expansion: 2006–2013

The German origin did not stay German for long. Beginning in 2006, the Austrian investment-fund community joined the initiative: **VÖIG** (*Vereinigung Österreichischer Investmentgesellschaften*), the **Oesterreichische Kontrollbank** (OeKB), and **ERSTE-SPARINVEST KAG** brought FundsXML into Austria and established the German-Austrian co-stewardship that still defines the standard's governance today. The chair of the FundsXML Standards Committee is to this day held by BVI, with VÖIG serving as deputy — an arrangement that reflects the order in which the two national communities joined rather than any weighting of their importance.

The pace of European adoption accelerated from there. In **2007**, **Robeco** carried FundsXML into the Netherlands. In **2008**, two further national communities joined simultaneously: **DIAMS** (*FundsXML France*) brought the standard into France, and the **Danish Investment Association IFR** adopted it under the *FundConnect* project for Denmark. The same year, **version 3.0 was published on 30 October 2008**, consolidating several rounds of extensions into a

single release. In **2009**, **La Banque Postale Asset Management** of Paris became the first major French adopter. In **2010**, the **AFG** (*Association Française de la Gestion Financière*, the French fund-industry association) endorsed the standard, and **Fundsquare** brought it into Luxembourg. **KNEIP**, also of Luxembourg, followed in **2011**, and **AXA France Investment Managers** in **2012**.

The decisive European moment came in **2013**, when **EFAMA** — the *European Fund and Asset Management Association* — recommended FundsXML as the standard format for the **Fund Processing Passport** (FPP), the pan-European framework for communicating the essential operational data of a fund to its distributors. EFAMA's endorsement moved FundsXML from "a widely-used industry schema" to "the European Fund Association's recommended format", and it placed the standard at the centre of cross-border fund distribution from that point onwards.

3.2.3 Regulatory Era and Version 4.x: 2014–2026

The third phase of FundsXML's history is the one most readers of this book will have lived through. Between 2014 and the present, the European fund industry was reshaped by a cascade of regulatory initiatives — MiFID II, PRIIPs, the SFDR and the broader sustainable-finance package, IDD, Solvency II — each of which demanded structured data of a kind that no legacy interface could provide. The FinDatEx consortium, supported by the major European industry associations, responded by publishing a series of templates: the *European MiFID Template* (EMT), the *European PRIIPs Template* (EPT), the *European ESG Template* (EET), the *European Feedback Template* (EFT), and the *Tripartite Template* (TPT) for Solvency II. These templates solved the semantic problem — *which fields must be exchanged and with what meaning* — but they said little about how the fields should travel between systems.

FundsXML became the natural carrier. The major architectural opportunity to make it so was the **4.0.0 release in 2017** — the first release since the original drafting that was deliberately *not* backward-compatible with earlier versions. Version 4.0 re-thought the top-level structure, introduced the separation of static and dynamic fund data that the standard still uses today, and cleaned up a number of historical compromises that had accumulated through the 3.x line. Subsequent 4.x releases then absorbed the FinDatEx templates one after another into the *RegulatoryReportings* area of the schema, so that a single FundsXML delivery could transport the classical master data, the portfolio, the transactions, and the regulatory templates in one envelope. This simple architectural decision — *same envelope, different freights* — is the reason FundsXML has become, in practice, the de-facto European inter-change format for structured fund data.

The 4.2 line, first released in May 2022 as version 4.2.0, consolidated this approach and added the connections to the European Single Access Point (ESAP) that Chapter 8 will discuss in detail. Eleven minor releases (4.2.0 through 4.2.10) have followed between 2022 and early 2026, each adding fields, tightening constraints, or tracking new FinDatEx template versions — without breaking the instance documents that earlier releases produced. The book you are holding describes the 4.2.x line and is accurate as of version 4.2.10.

Table 3.1 — Key Milestones in the History of FundsXML

Year	Event	Significance
2001	FundsXML created by Allianz Global Investment, COMINVEST, Credit Suisse Asset Management, DEKA, DWS, and Union Investment; early support from Vontobel, K&W Software AG, and FERI TRUST	Birth of the standard as a private inter-company schema among major German-speaking fund houses
2003	BVI adopts the standard; Version 1.0 published on 8 December 2003	Transition from a private schema to a named industry standard under German fund-association stewardship
2006	VÖIG, OeKB, and ERSTE-SPARINVEST bring FundsXML into Austria	Establishment of the German-Austrian co-stewardship that still defines governance today
2007	Robeco adopts FundsXML in the Netherlands	First expansion beyond German-speaking Europe
2008	DIAMS (<i>FundsXML France</i>) and the Danish Investment Association IFR (<i>FundConnect</i>) adopt the standard; Version 3.0 released on 30 October 2008	Simultaneous entry into France and Denmark; first major consolidating release
2009	La Banque Postale Asset Management adopts the standard	First major French asset-management adopter
2010	AFG endorses FundsXML; Fundsquare adopts it in Luxembourg	Endorsement by the French fund-industry association; entry into Luxembourg
2011	KNEIP adopts the standard in Luxembourg	Expansion within the Luxembourg fund-services ecosystem
2012	AXA France Investment Managers adopts the standard	Further French adoption
2013	EFAMA recommends FundsXML as the standard format for the Fund Processing Passport (FPP)	Recognition as the European Fund Association's recommended fund-data format
2017	FundsXML 4.0.0 released	First major architectural reset; deliberately not backward-compatible; introduces the static/dynamic separation
2022	FundsXML 4.2.0 released (4 May 2022)	Current major line; consolidated FinDatEx template integration and ESAP preparation
2026	Current 4.2.10 line	The version this book describes

Two final observations on the history are worth keeping in mind. First, FundsXML did *not* originate in Brussels. It was not imposed on the industry by a regulator, nor designed in a policy vacuum by a standards body with no operational stake. It grew out of the day-to-day practice of fund operations teams in six of Germany's largest asset-management houses in 2001, was picked up and formalised by their industry association (BVI) in 2003, and was extended into Austria, the Netherlands, France, Denmark, and Luxembourg over the following decade. This bottom-up genealogy explains a great deal about the standard's character: it is pragmatic, it is extensible, and it is, sometimes, a little idiosyncratic in places where a top-down design would have been tidier.

Second, the standard's evolution has been strictly additive for most of its recent history. Since the 4.0 reset, no subsequent release has broken existing instance documents in a way that required users to rewrite their producers or consumers; new modules have been added, old modules have been extended, and deprecated constructs have been retained alongside their replacements for several release cycles before removal. This conservative release discipline is another reason FundsXML has survived while less patient projects have not.

3.3 The Organisation Behind the Standard

A living standard needs a living organisation. The question every new FundsXML user eventually asks is *who decides?* — who decides what goes into the next release, who decides when that release is cut, and who the reader should contact if something in the schema is wrong. This section answers those questions in four short subsections.

3.3.1 The FundsXML Association

FundsXML is maintained by a registered, not-for-profit industry association whose members are drawn from every role in the European fund value chain: asset managers, fund administrators, depositary banks, data vendors, software houses, and a small number of consultancies. Membership is open, contributions are voluntary, and decisions are taken transparently within the working groups and formalised by the FundsXML Standards Committee (FSC). No single vendor or jurisdiction owns the standard, and no commercial license is required to implement it — both the schema and its documentation are freely available.

This governance model matters in practice. It means that a fund house in Warsaw or Helsinki can join a working group on equal terms with a depositary bank in Frankfurt; it means that a software vendor building a FundsXML parser does not need to negotiate redistribution rights; and it means that the standard's future direction is determined by the people who use it in production, rather than by a committee detached from the operational reality.

3.3.2 Working Groups and the Release Cycle

Strategic direction and maintenance of the standard are the responsibility of the **FundsXML Standards Committee (FSC)**, whose members are drawn from thirteen leading fund-industry organisations across Austria, Denmark, France, Germany, Luxembourg, the Netherlands, and Switzerland. The FSC decides on new memberships, creates and dissolves working groups, and — critically — decides on the release and realisation of working-group results. Day-to-day standardisation work is delegated to three dedicated working groups, summarised in Table 3.2.

Table 3.2 — FundsXML Working Groups

Working group	Scope	Typical outputs
Content & Technique	Manages the XML schema, defines data structures, ensures robustness and consistency of the standard	Schema releases, data-structure definitions
Promotion	Visibility and adoption of FundsXML across Europe through marketing, public relations, and member engagement	Conferences, outreach material, adoption campaigns
Documentation	Implementation guides, code examples, and schema references	Published guides, sample files, reference documentation

The release cycle follows the familiar semantic pattern. *Patch* releases within a 4.2.x line contain clarifications, documentation fixes, and strictly non-breaking adjustments — existing instance documents continue to validate unchanged. *Minor* releases (4.2, 4.3, and so on) add new elements or new modules but preserve backwards compatibility for anything that was already present. *Major* releases (the next of which would be 5.0) are reserved for architectural changes that cannot be made compatibly, and the association's policy is to avoid them unless no alternative exists — the 4.x line is now more than fifteen years old, which is a deliberate commitment rather than an accident.

The path a change takes from proposal to release is straightforward. Any member may raise a change request in the relevant working group. The Content & Technique working group discusses it, refines it, and — if the change is substantive — produces a draft schema update together with test files and a short rationale. The draft is then circulated to all members for review, adjusted in response to comments, and, if accepted by the FSC, scheduled into a release window. A release window is announced in advance so that implementers can plan their own update cycles; the release itself ships the new schema, a changelog, updated sample files, and any new or updated validation rules together as a single coordinated package.

3.3.3 Where the Standard Lives

FundsXML is, from a technical point of view, a small number of files: the XSD schemas, a collection of sample documents, a changelog, and supporting documentation. All of these are published openly. The authoritative home of the standard is the official website and the associated public source-code repository, where each release is tagged, signed, and accompanied by release notes. The issue tracker on the same repository is the canonical place to report bugs, request clarifications, or propose changes that are not yet mature enough for a working-group agenda.

This book deliberately does not print URLs in the running text; software and infrastructure change faster than books do. Appendix E collects the current set of official links, and the reader is encouraged to treat that appendix as the authoritative pointer whenever the present chapter says "the official site" or "the public repository".

3.3.4 Community and Events

Beyond the formal machinery of working groups and releases, FundsXML has a small but active community of practitioners who exchange knowledge at workshops, industry conferences, and informal meet-ups. Training material, webinars, and hands-on sessions are organised by the association itself and occasionally by member firms. For most readers of this book, the relevant point is simply that help exists and is reachable: a question asked in the right place usually receives a reasoned answer from someone who has already solved the same problem in production.

Taken together, the association, the working groups, the public repository, and the community form a lightweight but functional ecosystem. FundsXML is not a finished document to be consumed; it is a living project to which any serious implementer can, and occasionally should, contribute.

3.4 Design Principles

Before we turn to the schema itself, we pause to name the five principles that shape it. None of them is unique to FundsXML — every mature data standard wrestles with the same trade-offs — but the particular combination is characteristic, and recognising the principles in advance makes the structure of the schema considerably easier to read. These five principles reappear, explicitly or implicitly, in every chapter of Part II.

- 1. Separation of static and dynamic data.** FundsXML distinguishes the information about a fund that rarely changes — its name, its ISIN, its domicile, its investment policy — from the information that changes every valuation day — its NAV, its portfolio composition, its transaction flow. The two are carried in different substructures of the same Fund element, which allows a producer to ship a lean daily delta without re-transmitting the entire static record and allows a consumer to cache the static part between deliveries. This principle is visible in the schema as the pairing of *FundStaticData* and *FundDynamicData*, which Chapter 5 explores in detail.
- 2. Linkage by UniqueID, not by nesting.** A fund's portfolio may reference hundreds of distinct instruments, and a single instrument — say, a blue-chip European equity — may appear in the portfolios of many funds within the same delivery. Rather than repeat the full description of that instrument at every position where it occurs, FundsXML places each instrument once in a central *AssetMasterData* section and refers to it from every portfolio position by a *UniqueID*. The effect is to normalise the document, much as a relational database normalises a schema: the same information lives in one place and is referenced from wherever it is needed.
- 3. Extensibility through CustomAttributes.** No standard can enumerate every field that every member firm, in every jurisdiction, will ever need. FundsXML acknowledges this openly by providing a schema-sanctioned extension mechanism — the *CustomAttributes* element — that allows a producer to attach named key–value data to most structural elements without breaking the schema. National specialties, proprietary risk metrics, internal reference codes, and early experiments with new fields all live here without forcing a fork of the schema. Chapter 9 treats *CustomAttributes* in depth and explains how to use it responsibly.
- 4. Multi-language by design.** European fund distribution is inherently multilingual: the same fund may be sold in eleven countries under the same ISIN with eleven different marketing names and eleven versions of the same investment-policy description. FundsXML treats language as a first-class attribute of textual fields, so that a single delivery can carry names, classifications, and descriptions in several languages in parallel. There is no need to produce a separate file per language or to pick a canonical translation at the producer and hope the consumer accepts it.
- 5. Modular regulatory carriers.** EMT, EPT, EET, EFT, and TPT each have their own semantics, their own release cadence, and their own community. FundsXML does not attempt to rewrite them; it embeds them as self-contained modules within a *RegulatoryReportings* area, so that each can evolve on its own schedule while travelling in the same envelope as the core fund data. The envelope is stable; the freights are allowed to change independently.

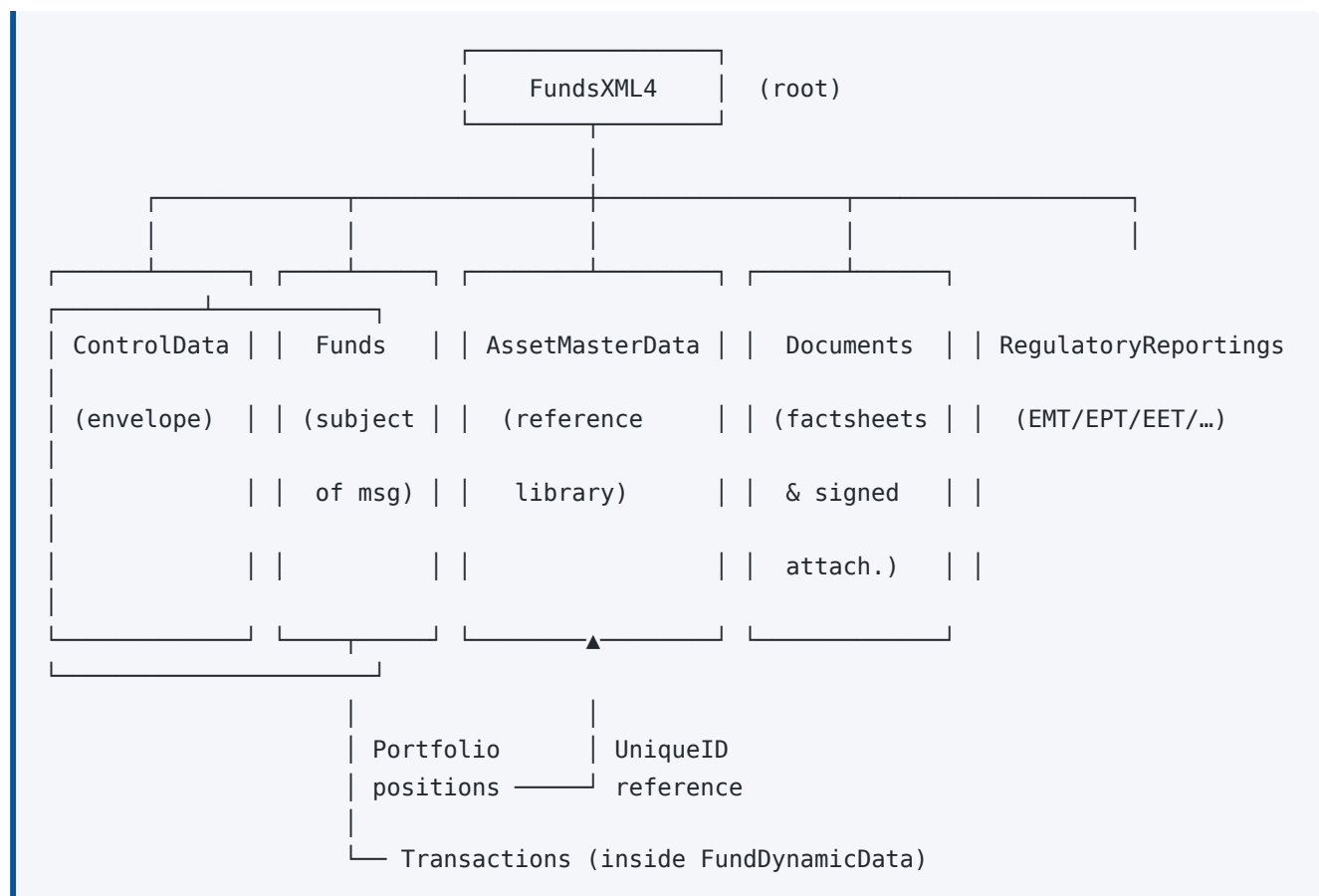
These five principles are the mental scaffolding on which the rest of the book rests. When, in a later chapter, a piece of the schema looks surprising, ask which of the principles it serves: nine times out of ten, the answer explains the design.

3.5 Schema Architecture at a Glance

We are now ready to look at the shape of FundsXML itself. This section is deliberately a *map*, not a tutorial. For each main area we give the purpose, the kind of content that lives inside it, the single most important idea to carry forward, and a forward pointer to the chapter of this book in which the area is treated in detail. No field listings, no code samples, no XPath expressions — those start in Chapter 4.

A FundsXML document has a single root element, conventionally named *FundsXML4*, and beneath that root live five main areas. These areas are the subject of Chapters 4 to 9. Figure 3.1 shows them schematically.

Figure 3.1 — The Five Main Areas of a FundsXML Document



The diagram makes two points worth stating in words. First, *Portfolios* and *Transactions* are not siblings of the five main areas; they live *inside* the Funds area, more precisely inside each Fund's *FundDynamicData*. Newcomers often expect Portfolio and Transactions as top-level root children and are briefly disoriented when they are not — remembering their position inside Funds saves a surprising amount of confusion later. Second, the arrow from Portfolio positions up into AssetMasterData is the graphical form of design principle 2: a position does not carry its own asset description; it references one, by UniqueID, in the central library.

We now walk through each of the five areas in the order a reader encounters them in a real delivery.

3.5.1 ControlData — The Envelope

ControlData is the envelope of every FundsXML delivery. It carries the metadata *about* the message rather than the substance *within* it: a unique document identifier, the reporting date to which the delivery refers, the creation time-stamp, the identity of the data sender and the intended receiver, the language of the delivery, and — crucially — the *DataOperation* flag that tells the consumer whether the file represents an initial load, an update to a previous delivery, or a deletion. Every FundsXML document contains exactly one *ControlData* element, and every downstream system reads it first.

The importance of *ControlData* is disproportionate to its size. It is the piece that allows a consumer to recognise, route, de-duplicate, and sequence a delivery; without it, the same file arriving twice would be indistinguishable from a genuine update, and a delivery arriving out of order would be impossible to reconcile against earlier ones. Chapter 4 walks through *ControlData* field by field and builds a complete, schema-valid example for the Europa Growth Fund.

3.5.2 Funds — The Subject of the Message

Funds is the area that most readers think of as "the FundsXML content proper". It contains one or more *Fund* elements, each representing a single legal fund or sub-fund, and within each *Fund* the structure splits — following design principle 1 — into *FundStaticData* and *FundDynamicData*. The static part describes the fund's identity: names in multiple languages, identifiers (ISIN, WKN, LEI, Valor), domicile, legal form, share-class structure, investment policy, benchmark, fees. The dynamic part describes the fund on a given valuation date: NAVs per share class, total net assets, performance figures, portfolio composition, and the day's transactions.

Share classes deserve a word here because they often cause the most friction for newcomers. A single fund in FundsXML may carry several share classes, each with its own ISIN, its own currency, its own distribution policy, and its own NAV; the fund's portfolio is typically shared across all share classes, while some of the dynamic figures are per-share-class. Chapter 5 is devoted entirely to the Funds area and uses the Europa Growth Fund, its three share classes, and their daily updates as a continuous example.

3.5.3 AssetMasterData — The Reference Library

AssetMasterData is the central reference library of a FundsXML delivery. It contains one entry per distinct instrument referenced anywhere in the delivery's portfolios, independent of how many times that instrument appears and in how many funds. Each entry gives the full static description of the instrument — asset class, identifiers, issuer, maturity where applicable, classification — and exposes a *UniqueID* by which portfolio positions can refer to it.

The value of *AssetMasterData* is exactly the value of normalisation in any database: the same information lives in one place, not many, and changes to an instrument's description need to be made once per delivery rather than hundreds of times per portfolio. The principle is important enough that Chapter 6 begins with a section on normalised referencing before it addresses individual asset classes.

3.5.4 Documents — Factsheets and Attachments

Documents is the area of the schema that most practitioners meet last, if at all, and yet it fills a surprisingly important role. It carries references to — or, in some deployments, the actual binary content of — fund-related documents that travel alongside the structured data: factsheets, prospectuses, key information documents (KIDs and KIIDs), sustainability-related disclosures, and audit reports. A *Document* entry names the document, classifies it, identifies the language and the effective date, and provides either a URL or an embedded representation.

The Documents area is also where FundsXML's support for XML digital signatures (XMLDSig) is rooted: a signed Document entry allows the producer to attest to the integrity and origin of an attached PDF in a way that survives transport through intermediate systems. Chapter 9 covers the Documents area together with signatures and country-specific additions.

3.5.5 RegulatoryReportings — The Regulatory Carriers

RegulatoryReportings is the area into which FundsXML absorbs the FinDatEx templates and related regulatory deliverables. It is organised as a container of self-contained modules, one per template: an *EMT* module for the European MiFID Template, an *EPT* for the PRIIPs template, an *EET* for the ESG template, an *EFT* for the Feedback template, and a *TPT* for the Solvency II Tripartite Template. Each module carries the fields that its template defines, with the same field names and the same data types, so that a FundsXML-aware consumer and a native template-aware consumer see the same data.

The architectural trick — and design principle 5 at work — is that FundsXML does not attempt to *replace* the templates. It carries them. When a template is revised by FinDatEx, the corresponding FundsXML module is updated in the next minor release without disturbing anything else in the delivery. Chapter 8 is the longest chapter in this book precisely because the regulatory carriers are where most of the day-to-day implementation effort goes.

A Summary Table

Table 3.3 — The Five Main Areas of FundsXML at a Glance

Area	Purpose in one sentence	Treated in
ControlData	Metadata of the delivery: who, when, for whom, which operation	Chapter 4
Funds	Static and dynamic fund data, share classes, portfolios, transactions	Chapters 5, 6, 7
AssetMasterData	Central reference library of instruments, addressed by UniqueID	Chapter 6
Documents	Factsheets, prospectuses, signed attachments	Chapter 9
RegulatoryReportings	EMT, EPT, EET, EFT, TPT as self-contained modules	Chapter 8

Table 3.3 is the single most important piece of Chapter 3 to keep within reach while reading the rest of the book. Whenever a later chapter refers to a "section" or a "module" of FundsXML, the table tells the reader where it sits in the overall picture.

3.6 A First FundsXML Skeleton

We close the architectural tour with the first real FundsXML fragment of the book. The listing below is a *skeleton*: it contains the element names and the hierarchy, but no values, no attributes beyond the namespace declaration, and no data beyond what is needed to show how the pieces fit together. No FundsXML validator would accept it, and no system would consume it. Its purpose is orientation, not execution.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">

  <ControlData>
    <!-- document id, reporting date, sender, receiver, operation -->
  </ControlData>

  <Funds>
    <Fund>
      <FundStaticData>
        <!-- names, identifiers, domicile, share classes (static) -->
      </FundStaticData>
      <FundDynamicData>
        <TotalAssetValues>
          <!-- NAVs and total net assets per share class -->
        </TotalAssetValues>
        <Portfolios>
          <Portfolio>
            <Positions>
              <Position>
                <!-- references AssetMasterData by UniqueID -->
              </Position>
            </Positions>
          </Portfolio>
        </Portfolios>
        <SingleFundFlows>
          <!-- transactions: subscriptions, redemptions, distributions -->
        </SingleFundFlows>
      </FundDynamicData>
    </Fund>
  </Funds>

  <AssetMasterData>
    <Asset>
      <!-- one entry per distinct instrument, addressed by UniqueID -->
    </Asset>
  </AssetMasterData>

  <Documents>
    <!-- factsheets, prospectuses, KIDs, signed attachments -->
  </Documents>
</FundsXML4>
```

```

</Documents>

<RegulatoryReportings>
  <!-- EMT / EPT / EET / EFT / TPT modules -->
</RegulatoryReportings>

</FundsXML4>

```

Even though the listing carries no real data, every design principle from §3.4 is visible in it. Principle 1 — the separation of static and dynamic data — appears as the pairing of *FundStaticData* and *FundDynamicData* inside the Fund element. Principle 2 — linkage by UniqueID — is the reason *Positions* holds only references and not instrument details, while the full instrument descriptions live once, in *AssetMasterData*, at the root level. Principle 3 — extensibility — is not printed above (we have left *CustomAttributes* out of the skeleton deliberately), but every major structural element in the real schema admits it. Principle 4 — multi-language support — is invisible in a skeleton without values, but the name and description fields inside *FundStaticData* will accept per-language variants when we fill them in. And principle 5 — modular regulatory carriers — is the reason *RegulatoryReportings* sits as its own top-level area alongside, rather than inside, the fund data it complements.

This is the last time in the book that a FundsXML sample will be this abstract. Beginning in Chapter 4, every listing carries real data for the Europa Growth Fund. Readers should note that the code listings in Chapters 4 through 8 are **illustrative**: element names, attribute values, and the precise field ordering are written for pedagogical clarity and do not always match the production schema down to the last character. Chapter 9 changes the rule and shows fully **schema-validated** examples against the current FundsXML 4.2.8 XSD, together with the `xmllint` commands that verify them. The complete, end-to-end example for the Europa Growth Fund is printed in Appendix D.

3.7 Key Takeaways

- FundsXML is not a regulatory imposition but a bottom-up industry standard. It was created in 2001 by six major German-speaking fund-management houses, adopted by BVI in 2003 (version 1.0 published on 8 December 2003), extended into Austria in 2006 under VÖIG, and expanded across Europe over the following decade. Today it is the de-facto European interchange format for structured fund data.
- The standard is maintained by an open, not-for-profit association whose working groups make decisions transparently and whose release discipline has kept the 4.x line backwards-compatible for more than fifteen years.
- Five design principles shape every FundsXML message: the separation of static and dynamic data, linkage by UniqueID rather than nesting, extensibility through *CustomAttributes*, multi-language support as a first-class feature, and modular regulatory carriers inside a stable envelope.
- A FundsXML document has one root and five main areas: *ControlData* (the envelope), *Funds* (the subject of the message), *AssetMasterData* (the reference library), *Documents* (factsheets and signed attachments), and *RegulatoryReportings* (EMT, EPT, EET, EFT, and TPT).
- *Portfolios* and *Transactions* are not top-level areas in their own right; they live inside each Fund's *FundDynamicData*. Positions reference *AssetMasterData* entries by UniqueID rather than repeating instrument details in place.

- Table 3.3 maps each of the five main areas to the chapter of this book that treats it in depth, and the reader may usefully keep it within reach while reading Part II.

With the map of the standard now in hand, we can zoom in on the first of the five areas — the envelope that every FundsXML delivery begins with. Chapter 4 turns to *ControlData* and walks through every field that identifies, dates, and routes a delivery in production.



PART II — FUNDSXML IN DETAIL · CHAPTER 4

ControlData — The Metadata of a Delivery

Document ID, reporting date, sender, and operation type

4.1 Setting the Scene: Opening the Envelope

At the end of Chapter 3, the operations team at the Europa Growth Fund had a mental map of the five main areas of a FundsXML document. The most natural next question is not *"how do we describe the fund?"* — that is Chapter 5's subject — but rather *"how do we describe the delivery itself?"* Who sent it, when, on behalf of whom, for which reporting date, and — most importantly — *what kind of delivery is it?* A first-time load is not the same thing as an afternoon correction of a morning NAV; a regular monthly update is not the same thing as the retraction of a delivery that should never have gone out. FundsXML answers all of these questions in the small but disproportionately important structure that sits at the top of every document: *ControlData*.

Three reasons justify beginning Part II with the envelope rather than the fund itself. First, *ControlData* is the first element every consumer of a FundsXML file reads. A production pipeline that fails to understand *ControlData* never even gets to the fund data, let alone to the portfolios or regulatory modules. Second, *ControlData* is what makes a file identifiable, routable, and de-duplicable — without it, the same message arriving twice is indistinguishable from a genuine update, and a message arriving out of order is impossible to reconcile against what came before. Third, and most importantly, *ControlData* is where FundsXML takes its one explicit position on *time* — on how deliveries relate to one another across hours and days. That position is simple, deliberate, and not always what newcomers expect, and the six pages of §4.6 devoted to it are the conceptual core of this chapter.

By the end of this chapter, you should be able to:

- name every mandatory field of a *ControlData* block and explain what it identifies;
- produce a *UniqueDocumentID* that is unique across a producer's output stream for the life of the system;
- distinguish *INITIAL*, *AMEND*, and *DELETE* deliveries and choose the right one for a given operational situation;
- trace a chain of related deliveries backwards to the *INITIAL* that started it;
- recognise the half-dozen mistakes that cause the most production incidents in *ControlData* handling.

No prior knowledge of the FundsXML schema beyond Chapter 3 is assumed. The chapter proceeds from purpose to fields to operations: §4.2 describes what *ControlData* *does* in a live pipeline; §4.3 gives the field skeleton; §4.4 and §4.5 walk through the identification and addressing fields; §4.6 covers *DataOperation* and its consequences; §4.7 treats the inter-document references that bind a stream together; §4.8 assembles everything into a complete, schema-valid example for the Europa Growth Fund. A short pitfalls list and the key takeaways close the chapter.

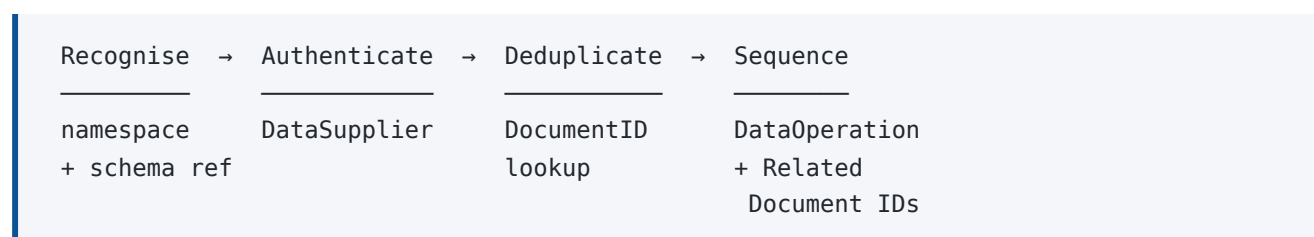
4.2 The Role of ControlData in a Production Flow

Before we look at any field, it is worth asking what a real consumer *does* with an arriving FundsXML file in its first few milliseconds of existence. The answer turns out to be a short, ordered list, and every item on the list is answered by ControlData. Understanding the list first makes the fields that follow feel obvious rather than arbitrary.

Imagine the classic production setup at a distributor in Paris: an SFTP drop-box that receives FundsXML files from a dozen different asset managers, a dispatcher that triages every arriving file, and a family of downstream processors that each handle a particular kind of content. When a new file lands in the drop-box at 07:03 a.m. CET, the dispatcher does four things, strictly in this order:

1. **Recognise.** What kind of file is this, and which schema version does it claim to conform to? The root element and its namespace declaration answer both questions; `ControlData` is inspected immediately afterwards to confirm that the file is a well-structured FundsXML delivery and to read the language of its contents.
2. **Authenticate at the business level.** Who sent this? The `DataSupplier` block inside ControlData is compared against an allowlist of approved senders. Transport-level authentication (TLS, SFTP keys) has already happened at this point; the dispatcher is now checking whether the *declared* sender is one it expects to receive data from.
3. **Deduplicate.** Have I seen this delivery before? The `UniqueDocumentID` is looked up in a persistent store of previously processed identifiers. If it is already there, the file is logged and silently discarded; if not, it is recorded and passed on.
4. **Sequence.** How does this delivery relate to what I already have? `DataOperation` and the `RelatedDocumentIDs` references decide whether the file is accepted as new, applied as an update, treated as a correction, or used to retract earlier content.

Figure 4.1 — The first four things a consumer does with a FundsXML file



Every one of these four steps runs *before* a single Fund, Portfolio, or RegulatoryReporting element has been read. That is why ControlData is small but disproportionately important: a mistake in the envelope makes the two hundred thousand lines behind it undeliverable.

The rest of this chapter treats the same four steps in the same order. §4.4 fills in *Recognise* and *Deduplicate* (document identity and timestamps); §4.5 fills in *Authenticate* (sender and contact); §4.6 and §4.7 fill in *Sequence* (data operations and inter-document references). When we assemble the complete example in §4.8, the four steps above are the checklist against which the finished block can be read.

4.3 A First Look — Structure and Mandatory Fields

At its heart, a `ControlData` block is small. Reduced to its mandatory fields only, with generic placeholders in place of real values, it looks like this:

```
<ControlData>
  <UniqueDocumentID>...</UniqueDocumentID>
  <DocumentGenerated>...</DocumentGenerated>
  <ContentDate>...</ContentDate>
  <DataSupplier>...</DataSupplier>
</ControlData>
```

Four elements — that is the whole of the mandatory core. Table 4.1 summarises every `ControlData` field at a glance.

Table 4.1 — `ControlData` fields at a glance

Field	Required	Type	Purpose
Version	no	string	Schema version the producer built against
UniqueDocumentID	yes	string	Identifies the delivery uniquely in the producer's output stream
DocumentGenerated	yes	dateTime (UTC)	Technical timestamp at which the file was created
ContentDate	yes	date	Business reporting date to which the content refers
DataSupplier	yes	complex	Identifies the business sender of the delivery
DataOperation	no	enumeration	Classifies the delivery: INITIAL, AMEND, DELETE
RelatedDocumentIDs	no	complex	Links the delivery to predecessors it depends on or supersedes
Language	no	ISO 639-1 code	Language of the textual content of the delivery

Several other fields are available but optional, and will appear in the sections that follow. The most important of them are *DataOperation*, which classifies the delivery as initial, amendment, or deletion; *RelatedDocumentIDs*, which links the current delivery to the ones it depends on or supersedes; *Language*, which declares the language of the textual content; and *Contact* (inside *DataSupplier*), which records the team or alias to call if the delivery turns out to be broken. §4.5 and §4.7 treat these in turn.

One small but frequent source of confusion is worth mentioning now: the order in which the fields appear inside a real `ControlData` block is prescribed by the schema, and it is *not* the didactic order of this chapter. A validator will reject a file in which, for example, `ContentDate` appears before `DocumentGenerated` even if every value is

correct. We treat the fields in the order that makes them easiest to understand; when we assemble the complete example in §4.8 we will rearrange them into the order the schema expects. The XSD in Appendix C is the authoritative reference for the exact sequence.

4.4 Identifying a Delivery — DocumentID, Timestamps, ContentDate

The first three mandatory fields answer the question *which delivery is this, and what does it refer to?* They look innocuous, but each of them carries a subtle design assumption that, if misunderstood, causes a surprising share of production incidents. We take them one at a time.

4.4.1 UniqueDocumentID — What Makes It Unique

The *UniqueDocumentID* is a string that identifies this particular FundsXML delivery. The schema places no structural constraint on the string other than that it exists; it is therefore tempting to treat the field as "some kind of identifier, the producer's choice". That reading is technically correct and operationally dangerous.

The guarantee a producer makes when it fills in *UniqueDocumentID* is stronger than XML validity. It is a *promise*, to every current and future consumer, that this exact value will never again appear on any other FundsXML delivery coming out of the producer's systems — not tomorrow, not next year, not after a system migration, not after a disaster-recovery restore from a six-month-old backup. The reason the promise matters is step 4 of §4.2: the consumer uses the *UniqueDocumentID* to recognise duplicates. If the producer reuses an identifier, even once, the consumer's deduplication logic either silently drops a real new delivery (thinking it is a duplicate) or silently accepts two different deliveries as one (corrupting its downstream data).

In practice, there are two sensible strategies.

The first, and the one we recommend as the default, is to use a random *UUID* — version 4 or the newer version 7. UUIDs are long enough that accidental collisions are astronomically unlikely, and they are trivial to generate in every programming language the fund industry uses. Version 7, which embeds a timestamp in the high bits, has the pleasant side effect that identifiers sort chronologically, which helps when browsing archives.

The second strategy is a *semantic* identifier constructed from known fields — something like `EGF-20260331-INITIAL-001`. Semantic identifiers are human-readable and sometimes useful during debugging. They are also fragile: the moment an operational situation demands that the same business day be processed twice (a restarted run, a re-export after a configuration fix), the naïve template collides with itself and the producer must invent an ad-hoc suffix. If you use a semantic scheme, build in a disambiguation suffix from the start — a sequence number, a UUID tail, or a timestamp to millisecond precision. Do not rely on "this can't happen twice".

Why not the filename? A natural shortcut is to reuse the filename of the FundsXML file as its *UniqueDocumentID*. This is almost always a mistake. Filenames are renamed by transfer agents, case-folded by Windows file systems, rewritten by archive extractors, and silently truncated by legacy pipelines. The *UniqueDocumentID* is part of the document; it survives every transport. The filename is a property of the envelope around the document; it does not.

The *UniqueDocumentID* is also *not* the place to carry human-readable metadata such as the fund name, the reporting date, or the operation type. Those have their own fields. Loading them into the identifier tempts downstream consumers to parse the identifier instead of reading the proper fields — and parsing identifiers across producers is a well-known source of inter-operation bugs.

4.4.2 DocumentGenerated and ContentDate — Two Different Clocks

The most common mistake in ControlData handling is to confuse the two timestamps. The two fields look similar, they occupy adjacent positions in the schema, and under normal circumstances they carry values close to each other. They mean completely different things.

DocumentGenerated is a *technical* timestamp. It records the moment at which the FundsXML file was produced by the sending system — the point at which the serialiser finished writing bytes to disk or to a network socket. It is the clock that operations, monitoring, and audit pipelines care about. Its type is `xs:dateTime`, and it is almost always expressed in UTC with a `Z` suffix.

ContentDate is a *business* timestamp. It records the reporting date to which the content of the delivery refers — the day whose NAV is being reported, the day on which the portfolio was valued, the day on which the regulatory template is effective. Its type is `xs:date` (no time component), and it is interpreted in the business calendar of the subject of the delivery, not in the time zone of the producing system.

An example makes the difference concrete. The Europa Growth Fund closes its books for 31 March 2026 at its Luxembourg valuation point (13:00 CET on that same day). The fund administrator's batch job runs overnight and produces the monthly FundsXML delivery at 06:47 UTC on 1 April 2026. The file is placed on the SFTP drop the moment it is finished. The correct ControlData values are:

```
<ContentDate>2026-03-31</ContentDate>
<DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
```

Any other combination is wrong. Setting *ContentDate* to `2026-04-01` would claim that the content describes 1 April, which is untrue. Setting *DocumentGenerated* to `2026-03-31T...` would lie about when the file came into existence.

A word on time zones. *DocumentGenerated* should be expressed in UTC. XML Schema allows local time with an offset (`2026-04-01T08:47:13+02:00`), and the schema accepts such values, but international *lieferbeziehungen* are better served by the Z convention: it avoids every ambiguity and simplifies comparison across files from different producers. If your system has a strong reason to emit local time with an offset, make sure the offset is correct and constant across daylight-saving transitions, and make sure every consumer you talk to can parse it.

4.4.3 Practical Recipes

The following short recipes cover the four situations that account for almost every real-world ControlData creation.

- **Daily NAV, standard business day.** *ContentDate* = today's valuation date. *DocumentGenerated* = the UTC moment the batch finished. *UniqueDocumentID* = a fresh UUID.

- **Month-end delivery.** *ContentDate* = the last business day of the month (not the first of the next month, and not a calendar month-end that fell on a weekend). *DocumentGenerated* = whenever the month-end batch actually ran, which may be a day or two later. *UniqueDocumentID* = a fresh UUID.
- **Ad-hoc delivery outside the regular cycle.** *ContentDate* = the business date the content refers to, even if no regular delivery exists for that date. *DocumentGenerated* = now. *UniqueDocumentID* = a fresh UUID.
- **Backfill for a missed day.** *ContentDate* = the day that was originally missed, possibly weeks in the past. *DocumentGenerated* = the UTC moment the backfill run finished. *UniqueDocumentID* = a fresh UUID — it is *not* the identifier of the original missed delivery, because that delivery never existed. §4.6 covers how backfills interact with *DataOperation*.

4.5 Sender and Contact — DataSupplier

Once a consumer has identified *which* delivery is on the wire, its next question is *who is talking*. FundsXML answers this with the mandatory *DataSupplier* structure, which names the business sender and optionally includes a *Contact* substructure for escalation.

4.5.1 DataSupplier — Who Is Sending, With What Authority

The *DataSupplier* block is a small record that describes the business entity responsible for the delivery. Its principal fields are *SystemCountry* (the ISO country code of the jurisdiction in which the supplier operates), *Short* (a compact code used for allowlist matching), *Name* (the full legal or commercial name), *Type* (an enumeration indicating the supplier's role — asset manager, fund administrator, depositary, and so on), and an optional *Contact* substructure (telephone or e-mail for incident escalation).

Table 4.2 — DataSupplier fields in practice

Field	Required	Used by consumer for	Typical value
SystemCountry	yes	Jurisdictional context	"LU"
Short	yes	Allowlist matching, compact display	"EAM"
Name	yes	Human display, audit logs	"Europa Asset Management S.A."
Type	yes	Categorisation of sender role	"AssetManager"
Contact	no	Incident escalation (Name, Phone, Email)	email or phone alias

The operational rule that matters more than any other is that a consumer that needs to match a sender against an allowlist matches on the *SystemCountry* / *Short* pair, not on the *Name*. Names change — a corporate rebrand, a merger, a translation of a German legal suffix into its English form — and consumers that match on the name will

silently start rejecting valid deliveries after each such change. The *SystemCountry / Short* pair is stable across most corporate events and is under the producer's control. Any fund-data consumer with more than a handful of counter-parties should treat the code pair as the primary key.

It is worth drawing a distinction that causes friction in complex delivery setups: the supplier is not necessarily the party that technically produced the file. A typical arrangement is that an asset manager contracts a fund administrator, who in turn runs the batch that emits the FundsXML document. In this arrangement, *DataSupplier* usually names the asset manager — the *business* originator of the content — rather than the administrator, which is merely the technical producer. Consumers authenticate against the asset manager because that is the party they have a contractual relationship with. The administrator's involvement is operational, not declarative. Chapter 13 returns to this distinction when it discusses implementation-project governance; for Chapter 4, the point is simply that "sender" and "producer" can legitimately be different, and *ControlData* models the sender.

4.5.2 Contact Information

Contact, which appears inside *DataSupplier*, records a telephone number, an e-mail address, or a team alias that the receiver can use if the delivery is broken and someone needs to be called. The content should be an *alias*, never an individual: the person who owns the inbox may change, but the alias survives. "fundsxml-support@europa-asset-management.com" is a good contact; "john.smith@europa-asset-management.com" is not.

4.6 DataOperation — Thinking Across Deliveries

We now arrive at the conceptual core of the chapter. The previous sections treated each delivery as an isolated object: it has an identity, a date, a sender, a receiver. In reality, every FundsXML delivery is part of a *stream* — a sequence of deliveries from the same producer to the same consumer, about the same fund, over time. *DataOperation* is the single field that FundsXML devotes to describing how one delivery relates to the ones that came before it. Six pages, the longest section of the chapter, are devoted to it, because it is also the single most common source of production incidents in FundsXML handling.

4.6.1 Why a Single Delivery Is Never the Whole Story

A live FundsXML interface is not a one-shot exchange. A fund administrator does not send a single file and never another one; it sends a daily NAV file every business day, a monthly portfolio file on the first working day of each month, an occasional correction when a mistake is discovered, and the occasional retraction when a file was emitted in error. Over a year, the same administrator–distributor pair exchanges hundreds of files, all referring to the same handful of funds, all sharing the same *DataSupplier*, and each of which changes the consumer's state in a specific way. The consumer's job is to reconstruct, at any given moment, the authoritative current picture of each fund — the most recent NAV, the most recent portfolio, the most recent regulatory template — from the accumulated history of deliveries.

FundsXML makes a deliberate decision about who manages that accumulation: *not the schema*. There is no schema-level notion of a session, a sequence number, a delivery counter, a heartbeat, or a replay acknowledgement. The schema does not tell the producer how often to send, and it does not tell the consumer what to do if a delivery is

missing. Everything it provides is one enumerated field — *DataOperation* — and a set of optional backward references — *RelatedDocumentIDs* — that §4.7 treats separately.

This minimalism is intentional. FundsXML is a document format, not a workflow protocol. It describes *what is in a delivery*, not *how deliveries are orchestrated*. Orchestration — scheduling, retry, acknowledgement, exactly-once handling — lives at the transport and application layers, and is the subject of Chapters 12 and 13. What the schema gives us is a shared vocabulary for saying, about each individual file, *what kind of delivery this is*. The six pages that follow are about using that vocabulary correctly.

4.6.2 The Three Operation Types

DataOperation is an enumerated field with exactly three values: **INITIAL**, **AMEND**, and **DELETE**. Each value carries a precise semantic meaning that every conforming consumer is expected to implement in the same way. Table 4.3 summarises the three at a glance; the narrative below it explains each in turn.

Table 4.3 — DataOperation semantics at a glance

Operation	Requires a previous delivery?	Consumer action	Typical trigger
INITIAL	no	Replace any existing data for the same subject and ContentDate	First delivery of a new reporting period
AMEND	yes	Modify the previously delivered data (extend, refine, or replace)	Late-arriving data, or a mistake detected after acceptance
DELETE	yes	Retract the previously delivered data as if it had never existed	A delivery was emitted in error

INITIAL marks the beginning of a stream, or the beginning of the data for a particular *ContentDate* within a stream. When a consumer receives an *INITIAL* delivery, its expected action is to install the data as the authoritative state for that subject and date, replacing anything it may already hold for the same combination. Repeated *INITIAL* deliveries for the same *ContentDate* are permitted and, under the standard's convention, are idempotent: receiving the same *INITIAL* twice must produce the same end state as receiving it once, provided the content is the same. The deduplication logic of §4.2 exists precisely to let consumers discard the redundant second copy efficiently.

AMEND is FundsXML's single-value umbrella for every non-initial, non-deleting change to an earlier delivery. It covers two operationally distinct but schematically identical cases. The first case is **extension**: a late-arriving piece of data — a share class whose NAV was delayed by a corporate event, a regulatory template that could not be finalised until the close of a different market — is shipped as an addition to what the consumer already has. The second case is **replacement**: a figure in a previously delivered file turns out to have been wrong, and the producer ships the corrected value in a new file so that the consumer can overwrite the error. Some older conventions (and some non-FundsXML data standards) distinguish these cases through separate **UPDATE** and **CORRECTION** enum values, but FundsXML 4.2.8 folds them into a single **AMEND** and leaves the semantic distinction to producer-consumer convention. Producers that need to signal which case they are sending typically do so through a text-level marker — a field in *CustomAttributes*, a tag in the delivery filename, or an out-of-band note to the consumer — rather than through the schema enumeration itself.

The consumer's expected action on receiving an *AMEND* delivery is to modify its stored state for the referenced subject and *ContentDate* according to the semantic intent that the producer has signalled. A replacement *AMEND* overwrites the earlier values; an extension *AMEND* adds to them. Either way, the *RelatedDocumentIDs* field (§4.7) should point at the previous delivery in the stream, so that the consumer can chain the changes correctly.

DELETE is the rarest and the most consequential of the three. It tells the consumer to treat a previously delivered document as if it had never existed. The canonical use case is a delivery that should never have been sent — a wrong fund, a wrong date, a draft that was promoted to production by mistake. The consumer's expected action is to remove the affected data from its authoritative state. Because *DELETE* leaves a hole, it is almost always followed by a corresponding *INITIAL* or *AMEND* that restores the intended state. A *DELETE* that is not followed by a replacement leaves the consumer without data, which is sometimes genuinely what is wanted (a subscription that was never real, a phantom transaction) but more often is a sign that the producer's pipeline is being used defensively when an *AMEND* would be clearer.

4.6.3 Idempotency and Ordering

Two properties of a consumer's handling of a FundsXML stream matter so much that they deserve to be named explicitly: *idempotency* and *order tolerance*.

A consumer is **idempotent** if processing the same delivery twice produces the same end state as processing it once. Idempotency is not an optional refinement; it is the minimum contract a consumer owes its operational environment, because networks retry, pipelines restart, and drop-boxes occasionally hand the same file to two processing workers. The technical mechanism that makes idempotency achievable is the *UniqueDocumentID* from §4.4.1: a consumer that records every *UniqueDocumentID* it has ever successfully processed, and that rejects a second encounter as a duplicate, is idempotent by construction. This is why §4.4.1 spends more words on the identifier than its two lines of XML would seem to deserve.

A consumer is **order-tolerant** if it behaves correctly when deliveries arrive in an unexpected sequence. The ideal case is that they arrive in the order they were sent; the real case is that networks reorder, queues deliver out of order, and retries of an older file can arrive after a newer one has already been processed. FundsXML does not provide a monotonic sequence number — there is no field whose sole purpose is to say "this delivery is the seventeenth from me". Instead, order is reconstructed after the fact from three signals: *ContentDate* (which business date does the content refer to?), *DocumentGenerated* (when was the file physically created?), and *RelatedDocumentIDs* (which earlier delivery does this one refer to?). A good consumer uses all three: it sorts deliveries for the same subject by *ContentDate*, breaks ties within a *ContentDate* by *DocumentGenerated*, and validates the chain of references with *RelatedDocumentIDs*.

A short scenario illustrates why this matters. Suppose the producer emits an *INITIAL* for 31 March 2026 at 07:00 UTC, and ten minutes later emits an *AMEND* (intended as a correction of an erroneous NAV) for the same date at 07:10 UTC. The two files travel over different network paths, and the *AMEND* overtakes the *INITIAL*: the consumer sees the *AMEND* at 07:11 UTC and the *INITIAL* at 07:14 UTC. What should the consumer do?

Two strategies are both acceptable, and a well-built pipeline will implement one of them explicitly. The first strategy, *buffer and reorder*, holds the *AMEND* aside until its referenced *INITIAL* has arrived, then applies them in the correct order; this gives the cleanest audit trail. The second strategy, *apply and reconcile*, applies the *AMEND* immediately, effectively installing the corrected data on an empty base, and then, when the late *INITIAL* arrives, refuses to replace the already-corrected state with the older uncorrected data; this is simpler to implement but requires that the

consumer's state machine knows the difference between "no data" and "already corrected". Either way, the consumer must not blindly overwrite corrected data with older uncorrected data. The *DocumentGenerated* timestamp on each delivery is what makes either strategy possible.

4.6.4 Anti-Patterns

The following short list of anti-patterns accounts for the majority of FundsXML production incidents we have seen or heard of. Each item is a rule, followed by its one-line justification.

- **Do not send an AMEND before the INITIAL it refers to exists.** The consumer has no base on which to apply the change, and will either reject the file or, worse, silently install partial data.
- **Do not reuse the same UniqueDocumentID for an INITIAL and a later AMEND.** Idempotent deduplication depends on uniqueness; a collision makes the consumer drop the second file as a duplicate even though its content differs.
- **Do not send two INITIAL deliveries for the same ContentDate without a clear resolution rule.** The consumer does not know which one is authoritative. If the second *INITIAL* is meant to replace the first, make it an *AMEND*.
- **Do not use DELETE without a follow-up.** A *DELETE* leaves a hole in the consumer's state. If the intent is to correct rather than to retract, use *AMEND*. If the intent is genuinely to retract, make the retraction explicit in the consumer's audit trail, ideally by following it with an *INITIAL* that restores the intended state (which may be "no data at all for this date").
- **Do not mislabel a replacement AMEND as a silent delta.** Because FundsXML collapses extensions and replacements into a single *AMEND* enum value, the producer must communicate the intent clearly — through a filename convention, a *CustomAttributes* flag, or an out-of-band note. A consumer that cannot tell a replacement from an extension will either lose data or fail to propagate corrections.

4.6.5 A Small Example Series — ControlData Only

To see the three operations working together, consider four successive *ControlData* blocks for the same Europa Growth Fund over a four-day period. The rest of the file — *Funds*, *Portfolios*, *AssetMasterData* — is elided, because the payload is the subject of later chapters. Only the envelope is shown.

Day 1, the first delivery of the month-end:

```
<ControlData>
  <UniqueDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b001</UniqueDocumentID>
  <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>...</DataSupplier>
  <DataOperation>INITIAL</DataOperation>
  <Language>en</Language>
</ControlData>
```

Day 2, a late-arriving piece of data extends the previous delivery without replacing it:

```

<ControlData>
  <UniqueDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b002</UniqueDocumentID>
  <DocumentGenerated>2026-04-02T09:12:04Z</DocumentGenerated>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>...</DataSupplier>
  <DataOperation>AMEND</DataOperation>
  <RelatedDocumentIDs>
    <RelatedDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b001</RelatedDocumentID>
  </RelatedDocumentIDs>
  <Language>en</Language>
</ControlData>

```

Day 3, an NAV that was reported incorrectly on Day 1 is corrected:

```

<ControlData>
  <UniqueDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b003</UniqueDocumentID>
  <DocumentGenerated>2026-04-03T14:28:50Z</DocumentGenerated>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>...</DataSupplier>
  <DataOperation>AMEND</DataOperation>
  <RelatedDocumentIDs>
    <RelatedDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b002</RelatedDocumentID>
  </RelatedDocumentIDs>
  <Language>en</Language>
</ControlData>

```

Day 4, the producer discovers that the entire Day 1 delivery was emitted for the wrong fund and retracts it:

```

<ControlData>
  <UniqueDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b004</UniqueDocumentID>
  <DocumentGenerated>2026-04-04T08:03:22Z</DocumentGenerated>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>...</DataSupplier>
  <DataOperation>DELETE</DataOperation>
  <RelatedDocumentIDs>
    <RelatedDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b003</RelatedDocumentID>
  </RelatedDocumentIDs>
  <Language>en</Language>
</ControlData>

```

Several observations deserve to be made about this small series. First, every delivery has a *new UniqueDocumentID* — never a reused one, even though the subject and the *ContentDate* are the same throughout. Second, every non-*INITIAL* delivery populates *RelatedDocumentIDs* to point at the immediately preceding delivery in the stream, not at the *INITIAL* from Day 1. This is the linear-chain convention that §4.7 recommends: it makes the full history reconstructable by walking the chain backwards, and it avoids the question of what a *RelatedDocumentID* should mean

when it points at a delivery that has itself been superseded. Third, the *DataSupplier* block is identical across all four deliveries, because the business sender has not changed — only the operational relationship between the deliveries has.

Finally, a meta-observation: most of the complexity in the *DataOperation* model lives on the consumer side, not on the producer side. A disciplined producer can set *DataOperation* correctly from a short rule set — "new data for a new date is *INITIAL*, any change to existing data is *AMEND*, retraction is *DELETE*" — and signal the fine distinction between an extension-*AMEND* and a replacement-*AMEND* through filename conventions or producer-consumer-specific metadata. A consumer, by contrast, must cope with every legal sequence the producer universe can throw at it, including ones the original producers did not anticipate. When in doubt, err on the side of the consumer: design *DataOperation* usage to make the consumer's life predictable, not the producer's life convenient.

4.7 Versioning and Related Documents

Where §4.6 treated the *kind* of a delivery, this section treats the *links* between deliveries. Two topics belong here: the *RelatedDocumentIDs* element, which binds non-*INITIAL* deliveries to the ones they depend on, and the schema version declared inside `ControlData`, which ties every delivery to a specific release of the FundsXML standard.

4.7.1 RelatedDocumentIDs — The Backward Pointer

RelatedDocumentIDs is an optional structure that, when present, lists one or more *UniqueDocumentID* values referring to earlier deliveries. It appears in the *AMEND* and *DELETE* cases shown in §4.6.5 and is conventionally absent from *INITIAL* deliveries, which by definition do not refer to anything earlier. The schema even makes *RelatedDocumentIDs* mandatory on a *DELETE* operation — a *DELETE* that does not explicitly name the delivery being retracted is not a legal FundsXML file.

Three properties of *RelatedDocumentIDs* are worth naming explicitly. First, the pointer is *backward*: an amendment points at the delivery it changes, not the other way round. There is no forward "this-delivery-has-been-superseded-by" pointer in the schema. A consumer that wants to know "is this delivery still authoritative?" must either remember the answer as it processes each new arrival or walk the chain backward from the newest delivery. Second, the pointer is *informative*, not enforced at the schema level: the XSD does not check that the referenced *UniqueDocumentID* actually exists, that it belongs to the same subject, or that the referencing operation makes semantic sense against what was referenced. All of that is a business-rule concern, and Chapter 10 treats it in its discussion of two-stage validation. Third, the element is *potentially transitive*: a second *AMEND* can refer to the first *AMEND*, forming a chain, or it can refer directly back to the original *INITIAL*, skipping the intermediate step. Both conventions are seen in the wild, and neither is forbidden by the schema.

Because both conventions are possible, producers and consumers must agree on one. The convention we recommend — and that Chapter 13 will formalise as a project-level decision — is the *linear chain*: every non-*INITIAL* delivery refers to the immediately preceding delivery in the stream, regardless of operation type. The result is a singly-linked list stretching from the newest delivery back to the *INITIAL*; walking the list backwards always reconstructs the full history, and every intermediate step is explicitly represented. The alternative convention — always pointing at the *INITIAL* — is simpler to construct but hides intermediate steps and makes it harder to tell whether a replacement *AMEND* was applied before or after an extension *AMEND*. Pick one, document it, and stick to it.

One final note: *RelatedDocumentIDs* is plural because there are cases in which one delivery legitimately depends on more than one predecessor. The most common is a consolidated correction that replaces two or more earlier deliveries at once. These cases are rare enough that most implementations can treat the element as effectively singular, but the plural form is there when it is needed.

4.7.2 Schema Version Handling

A point of practical importance that often surprises first-time FundsXML implementers: the schema file does **not** declare a `targetNamespace`. This means instance documents live in the unnamed namespace, and the root element is written without a default namespace declaration. Instead, the root element carries an `xsi:noNamespaceSchemaLocation` attribute that points directly to the schema file for the version the producer is building against:

```
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="https://github.com/fundxml/schema/releases/download/4.2
.2/FundsXML.xsd">
```

Every released version of the schema is published as a GitHub release under the `fundxml/schema` repository. The URL follows a predictable pattern — `https://github.com/fundxml/schema/releases/download/{version}/FundsXML.xsd` — where `{version}` is the release tag (e.g. 4.2.2). This gives every instance document a stable, publicly resolvable reference to the exact schema it was validated against, and it gives any consumer or validator a direct download path to that schema without needing a local copy.

Upgrading to a newer schema version is correspondingly straightforward: the producer updates the version number in the URL. Moving from 4.2.2 to 4.2.5, for example, requires nothing more than changing the root element to:

```
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="https://github.com/fundxml/schema/releases/download/4.2
.5/FundsXML.xsd">
```

All versions of FundsXML since 4.0.0 are designed to be backward compatible, so within the 4.x line such an upgrade is a smooth transition: a consumer built against 4.2.2 will usually accept files produced against 4.2.5, and vice versa. The backwards-compatibility discipline described in §3.3.2 ensures that *patch* releases add clarifications and non-breaking adjustments, while *minor* releases add new elements or modules but preserve compatibility for everything that was already present.

In addition to the schema reference in the root element, the schema version is also communicated *inside* the document, in `ControlData/Version`. A consumer can inspect this value before running schema validation to confirm which version the producer intended. Recording the version in the audit log alongside the delivery is a discipline worth following: mismatches between producer and consumer schema versions are a recurring source of "it worked yesterday, what changed?" incidents, and the audit log is often the only way to reconstruct what happened.

4.8 A Complete ControlData Block for the Europa Growth Fund

We are now ready to assemble the first complete, schema-valid example of the book. The block below is the actual *ControlData* envelope of the Europa Growth Fund's month-end delivery for 31 March 2026 — the same delivery from which every later chapter of Part II will show a fragment. Field order follows the sequence the schema requires, not the didactic order of this chapter.

```
<ControlData>
  <UniqueDocumentID>8a1c0e76-1b7d-4f55-9d2e-11f4a0c8b001</UniqueDocumentID>
  <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>
    <SystemCountry>LU</SystemCountry>
    <Short>EAM</Short>
    <Name>Europa Asset Management S.A.</Name>
    <Type>AssetManager</Type>
    <Contact>
      <Email>fundsxml-support@europa-asset-management.com</Email>
    </Contact>
  </DataSupplier>
  <DataOperation>INITIAL</DataOperation>
  <Language>en</Language>
</ControlData>
```

Reading the block field by field, every design choice from the previous sections becomes visible. The *UniqueDocumentID* is a UUID, unique across the producer's output stream for the life of the system (§4.4.1). The *DocumentGenerated* timestamp is expressed in UTC and records the moment the batch job finished writing the file; the *ContentDate* is a plain date and refers to the business reporting day, which is the last business day of March 2026 (§4.4.2). The *DataSupplier* block names the fund's asset manager rather than the fund administrator that technically produced the file, and carries the *SystemCountry* / *Short* pair that a consumer's allowlist will match against rather than the *Name* (§4.5.1). The *Language* is English, which indicates the language of the textual content inside the file — the fund's *marketing* names and descriptions may still appear in several languages, and Chapter 5 treats that case explicitly. The *DataOperation* is *INITIAL*, because this is the first month-end delivery for 31 March 2026; the next delivery in the same stream, whatever it turns out to be, will populate *RelatedDocumentIDs* to point back at this file's *UniqueDocumentID* (§4.6, §4.7).

This block is short, and yet every one of the four steps from §4.2 is now answerable from it alone. A consumer that reads these lines and stops has enough information to decide whether to continue parsing the rest of the file, and in production that is exactly what the first stage of a real pipeline does. The complete end-to-end example of the Europa Growth Fund's month-end delivery — this *ControlData* block plus the *Funds*, *AssetMasterData*, *Documents*, and *RegulatoryReportings* sections — is printed in Appendix D, and each of the remaining chapters of Part II will take one piece of it for detailed treatment.

4.9 Common Pitfalls

The following short list captures the half-dozen mistakes that, in our experience, account for the majority of ControlData-related production incidents. Each entry gives the symptom, the underlying cause, and the fix.

- **A new delivery is silently dropped as a duplicate.** *UniqueDocumentID* has been reused, usually because it was generated from a non-unique source such as the filename or a timestamp at second precision. Switch to UUIDs.
 - **Reports appear for the wrong day.** *ContentDate* has been set to the day the file was produced rather than to the business reporting date. Re-read §4.4.2 and fix the producer's timestamp derivation logic.
 - **The consumer rejects a known sender as unknown.** The allowlist is matching on *Name* rather than on the *SystemCountry / Short* pair, and the sender's legal name has just changed (merger, rebrand, translation). Switch the match to the stable code pair.
 - **An AMEND is rejected because the consumer sees no base state.** The *AMEND* was sent before its *INITIAL* was successfully processed, or the *RelatedDocumentIDs* pointer is missing or refers to a non-existent document. Re-issue the *INITIAL* first, or fix the pointer.
 - **A correction reaches investors without triggering recalculation.** A replacement *AMEND* was emitted without the producer-consumer convention that distinguishes replacement from extension. The consumer's audit hooks assumed the *AMEND* was an additive extension rather than a correction and silently installed the new figures without re-issuing affected downstream documents. Agree on a replacement/extension signal between producer and consumer before going live.
 - **A retracted delivery leaves a hole in the consumer's state.** A *DELETE* was sent without a follow-up *INITIAL* restoring the intended state. Either send the restoring *INITIAL*, or confirm explicitly that an empty state is intended and document the gap.
-

4.10 Key Takeaways

- ControlData is the envelope of every FundsXML delivery. It identifies, authenticates, deduplicates, and sequences the file — and all of this happens before any of the fund data behind it is read.
- Four mandatory fields form the core: *UniqueDocumentID*, *DocumentGenerated*, *ContentDate*, and *DataSupplier*. The optional *DataOperation*, *RelatedDocumentIDs*, and *Language* fields are essential in any production setup.
- *DocumentGenerated* (technical timestamp) and *ContentDate* (business reporting date) are two different clocks and must never be confused.
- *DataSupplier* is matched by its *SystemCountry / Short* pair, not by *Name*; an allowlist that matches on the name will silently break on every rebrand or translation.
- *DataOperation* — *INITIAL*, *AMEND*, *DELETE* — is the only bridge FundsXML provides between successive deliveries. The schema collapses both extensions and replacements into a single *AMEND* value; the producer-consumer relationship must carry the finer distinction through convention. Idempotency and order tolerance on the consumer side are what make the bridge usable in practice.
- *RelatedDocumentIDs* carries backward pointers; the recommended convention is a linear chain, in which every non-*INITIAL* delivery refers to the immediately preceding delivery in the same stream.

- The complete ControlData block in §4.8 is the first schema-valid FundsXML fragment of Part II, and the same values reappear in every subsequent example through to the end of the book.

With the envelope in place, we can now open it and look inside. Chapter 5 turns to the subject of every FundsXML message — the fund itself — and to the static and dynamic data that describe it.



Funds, Sub-Funds, and Share Classes

Static and dynamic fund information

5.1 Setting the Scene: Inside the Envelope

At the end of Chapter 4 we had a `ControlData` block for the Europa Growth Fund's month-end delivery on 31 March 2026. We knew who had sent it, whom it was meant for, which day it referred to, and that it was the first delivery of its stream. What we did not know was anything about the fund itself. This chapter fills that gap. It is the longest chapter in the book — forty pages — because the `Fund` element is the element about which everything else in a FundsXML delivery eventually revolves, and because it is also the element where newcomers most often get lost.

Before we begin, one pedagogical decision deserves to be stated up front. The Europa Growth Fund, as we learned in Chapter 1, is a European UCITS equity fund distributed across eleven countries. In reality it has three distinct share classes: a retail euro-denominated accumulating class, a retail Swiss-franc-denominated currency-hedged accumulating class, and an institutional euro-denominated distributing class. Every field we discuss in this chapter exists at one of two levels: either at the level of the fund as a whole, or at the level of an individual share class. The two levels are easy to mix up, and a reader who has to track both at once while also learning about identifiers, classification, and fees is going to struggle.

We therefore take the chapter in four parts. **Part I** (§5.2 to §5.7) pretends the Europa Growth Fund has exactly one share class and treats every topic as though it lived cleanly on the fund element. **Part II** (§5.8 to §5.11) relaxes the pretence, puts the three share classes back, and walks through the topics on which the fund-level picture branches. **Part III** (§5.12 and §5.13) treats the dynamic data — NAVs, total net assets, performance figures — in the light of the share-class structure we have just built. **Part IV** (§5.14 to §5.17) covers umbrella funds, assembles the complete `Fund` element, lists the common pitfalls, and summarises the chapter. By the end of Part I the reader should feel comfortable reading a fund's identity; by the end of Part II the reader should be able to talk fluently about share classes; by the end of the chapter the reader should be able to read a complete `Fund` element on sight and know which of the remaining chapters of Part II of the book each fragment of it belongs to.

By the end of this chapter, you should be able to:

- identify which information in FundsXML belongs to a fund and which belongs to a share class;
- produce and populate every mandatory field of the `Fund` element for a realistic European UCITS;
- choose the right identifier for every jurisdiction in which a fund is distributed;
- model multilingual fund data without duplicating documents per language;
- distinguish fund-level, share-class-level, and regulatory-level representations of fees and risk indicators;
- read a complete `Fund` element with multiple share classes on sight and point at the pieces each later chapter of the book treats in depth.

5.2 The Fund Element in Context

Inside every FundsXML delivery, immediately after the ControlData envelope, sits a `<Funds>` container. The container is a simple list: it contains one or more `<Fund>` elements, each representing a single fund. The list is the schema's way of saying that one delivery may describe more than one fund at a time.

Two production patterns dominate in practice. The first pattern, **one fund per file**, is the one we have been assuming so far: the Europa Growth Fund is delivered in its own FundsXML file, the `<Funds>` container holds exactly one `<Fund>`, and the distributor's dispatcher routes the file by looking at the DataSupplier and the fund's ISIN. This pattern is common when the producer wants fine-grained control over who receives which fund — particularly where distribution agreements vary by country or share class. The second pattern, **several funds per file**, is more common in fund-administrator batches: a single file carries every fund the administrator serves for a given asset-management client, often several dozen at a time. The container form of `<Funds>` exists precisely so that the second pattern does not require a wrapper document of its own. Both patterns are valid FundsXML; the choice between them is a pipeline design decision, not a schema decision, and Chapter 12 will come back to it when it discusses integration into system landscapes.

Whichever pattern is used, each `<Fund>` element has the same shape. Four elements are mandatory and appear at the very top of every `<Fund>`; the rest follow the separation of static and dynamic data that we named in Chapter 3 as design principle 1, and are wrapped into an optional `<SingleFund>` container (for non-umbrella funds) or a `<Subfunds>` container (for umbrella structures, §5.14):

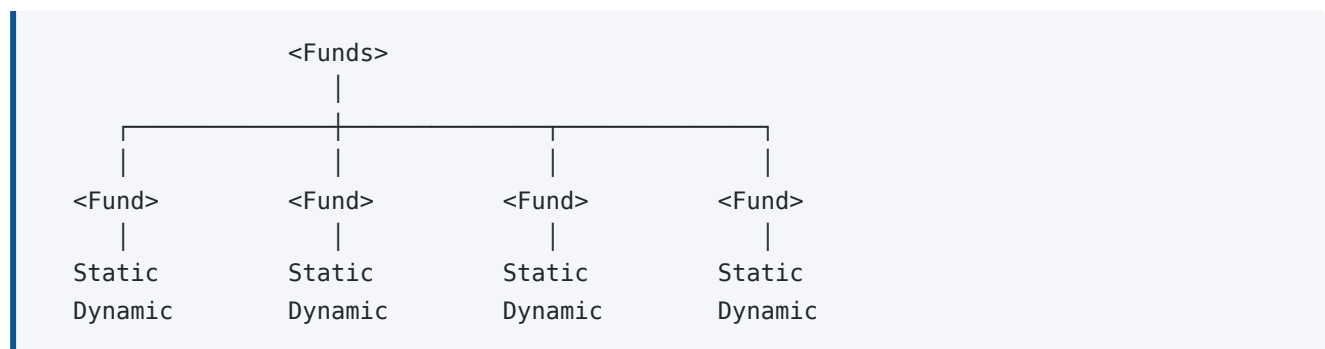
```
<Fund>
  <Identifiers>
    <!-- LEI of the fund and any other fund-level identifier -->
  </Identifiers>
  <Names>
    <!-- OfficialName, optionally MarketingName, ShortName, LanguageNames -->
  </Names>
  <Currency>EUR</Currency>
  <SingleFundFlag>true</SingleFundFlag>
  <FundStaticData>
    <!-- domicile, legal structure, classifications, benchmarks, ongoing costs, ... -->
  </FundStaticData>
  <FundDynamicData>
    <!-- fund-level total net assets, portfolio, fund-level benchmark values, ... -->
  </FundDynamicData>
  <SingleFund>
    <!-- SingleFundStaticData and the list of ShareClasses -->
  </SingleFund>
</Fund>
```

The shape deserves two observations. First, `Identifiers`, `Names`, `Currency`, and `SingleFundFlag` are **mandatory** and appear before the static/dynamic split. They establish the fund's identity before anything else; a `<Fund>` element that omits them is not schema-valid. Second, the static/dynamic split is not cosmetic. Many real pipelines ship `FundStaticData` only when something has changed — perhaps once a month, perhaps only when

a new share class is launched or a fee is revised — while shipping a fresh `FundDynamicData` block every valuation day. Doing so is entirely legal: both `FundStaticData` and `FundDynamicData` are optional, a delivery may contain one, the other, or both, and consumers are expected to maintain the last-known state of each half independently. The `DataOperation` semantics of Chapter 4 apply to whichever half is present.

The `<SingleFund>` wrapper is where the share-class structure lives. Newcomers often expect `<ShareClasses>` to hang off `FundStaticData`, but the schema puts them one level higher: `Fund/SingleFund/ShareClasses`, as a sibling of the static/dynamic blocks rather than a child of the static one. The reason is that `<SingleFund>` is one of two alternatives — the other being `<Subfunds>` for umbrella structures — and making the choice a first-class sibling keeps the two cases symmetric. We take `<SingleFund>` as the running case for the whole chapter and return to `<Subfunds>` in §5.14.

Figure 5.1 — Funds as a list



Each of the `<Fund>` boxes is structurally independent of the others: a consumer that cares only about one fund can read that fund and stop without loss of correctness. The `AssetMasterData` library (Chapter 6) lives outside the `<Funds>` container, as we saw in Chapter 3, so that instruments shared between several funds in the same delivery can be described once and referenced many times. That cross-fund sharing is the subject of Chapter 6, not this one; Chapter 5 treats each `<Fund>` as though it were the only fund in the file.

5.3 FundStaticData vs FundDynamicData Revisited

Chapter 3 introduced the static/dynamic split as design principle 1. Now that we are about to populate each half, it is worth pausing to ask a more operational question: *which field goes where?* The principle is intuitive in the limit — a fund's legal name is static, its NAV is dynamic — but the middle is less obvious. Is the *launch date* of the fund static or dynamic? It never changes, so it sounds static, but many reference systems store it alongside performance data for historical calculations, which makes it feel dynamic. Is the fund's *ongoing charges* figure static? It is published once a year and does not move between publications, so it is static by default, but a producer that recomputes it on demand at every delivery could reasonably place it in the dynamic half. FundsXML takes a position on each of these questions, and the position is worth understanding because it is the organising principle of every `Fund` element in the book.

The rule of thumb is straightforward. A field belongs in `FundStaticData` if, in the ordinary course of operating a fund, it would *not* need to be re-transmitted to downstream consumers on every valuation day. A field belongs in `FundDynamicData` if, in the ordinary course of operating a fund, it *would*. Table 5.1 summarises where the com-

moner fields land.

Table 5.1 — Where the commoner fields of a fund live

Mandatory header	FundStaticData	FundDynamicData	SingleFund / <ShareClass>
Identifiers/LEI (fund-level)	DomicileCountry	Fund-level TotalAssetValues	Per-class Identifiers (ISIN, GermanWKN, SwissValorenCode)
Names	ListedLegalStructure	Fund-level benchmark values	Per-class Names, Currency, ShareClassType
Currency	InceptionDate (fund)	Portfolio composition (→ Chapter 6)	SubscriptionRestrictions, CurrencyHedgedFlag
SingleFundFlag	FundTexts (strategy, risk, ...)		Prices and per-class TotalAssetValues
	Classifications		Fees, Distributions, Flows (→ Chapter 7)
	Benchmarks (static)		PerformanceFigures
	OngoingCosts		HighWatermark, InceptionDate (per class)
	SFDRProductType		RegistrationCountries

Three lines of the table need commentary. **Portfolio composition** lives inside `FundDynamicData/Portfolios` — that is the reason the table lists it in the third column — but it is a large enough topic to deserve its own chapter, and this chapter does not treat it. Chapter 6 walks through every asset class and every position-level field. **Transactions** (subscriptions, redemptions, distributions, corporate actions) live at the share-class level: subscriptions and redemptions under `ShareClass/Flows`, dividend events under `ShareClass/Distributions`. They are the subject of Chapter 7. Sections §5.12 and §5.13 of this chapter will cover per-class NAVs, fund-level totals, and performance figures, and will elide portfolio and flow detail when we get there.

A fourth observation is worth stating while the table is in view. **Per-class dynamic data lives on each class, not in FundDynamicData**. NAV per unit, shares outstanding, per-class total net assets, per-class performance, per-class cash flows and distributions, and per-class portfolios all live inside the individual `<ShareClass>` element under `SingleFund/ShareClasses`. `FundDynamicData/TotalAssetValues` carries a single aggregate figure for the whole fund, and `FundDynamicData/Portfolios` carries a shared fund-level portfolio where classes are backed by a common asset pool. The split is not always intuitive at first reading; §5.12 returns to it with concrete examples.

The operational consequence of the split is worth stating plainly. In a pipeline designed with static/dynamic separation in mind, the producer's batch job runs in two phases. The first phase, triggered only when something in the fund's identity changes, emits a `FundStaticData` update — a new share class was launched, a fee was revised, a benchmark was switched. The second phase, triggered on every valuation day, emits a `FundDynamicData` update — today's NAVs, today's shares outstanding, today's portfolio, today's transactions. The two streams flow through the same FundsXML documents, are sequenced by the same `DataOperation` mechanics from Chapter 4, and are reconciled by the consumer into a single current-state picture. It is perfectly legal under the schema to ship both halves in every delivery — and many producers do, because the redundancy is cheap and the operational simplification is real. But the ability to split them is available when it matters.

5.4 Fund Identifiers — ISIN, WKN, LEI, Valor, and Friends

The first substantive field group of the chapter is the one most likely to trip up a first-time implementer. FundsXML carries *several* identifiers for each fund, and the reason is that no single identifier universe covers all of the jurisdictions and all of the consumers that European fund data touches. A German investor needs a WKN. A Swiss investor needs a Valor. A regulator needs an LEI. A cross-border distribution agreement is written around ISINs. A legacy Bloomberg terminal may only accept its own proprietary ticker. All of these identifiers coexist, none of them is individually sufficient, and the schema supports all of them at once.

5.4.1 The Hierarchy of Identifiers

Identifiers in FundsXML live at two levels, and both levels use exactly the same container: a `<Identifiers>` element (of schema type `IdentifiersType`) that carries a fixed set of named children — `ISIN`, `Bloomberg`, `CUSIP`, `GermanWKN`, `LEI`, `MexID`, `ReutersRIC`, `SEDOL`, `SwissValorenCode`, `SwiftBIC` — plus a repeatable `OtherID` escape hatch for everything the schema does not name. One `<Identifiers>` block sits at the top of the `<Fund>` element and carries the fund-level identifiers; a separate `<Identifiers>` block sits at the top of each `<ShareClass>` element and carries the share-class-level identifiers. The two use the same vocabulary but populate different fields.

Some identifiers — principally the **LEI**, together with any regulator-issued fund number — identify the fund as a legal entity. These are *fund-level* identifiers; they exist once per fund and do not care how many share classes the fund has, and they sit inside the fund's own `<Identifiers>` container. Others — **ISIN**, **GermanWKN**, **SwissValorenCode**, **CUSIP**, **Bloomberg ticker** — identify a separately investable unit, which in practice means a single share class. These are *share-class-level* identifiers; a fund with three share classes has three `<Identifiers>` blocks, one per class, each populated with the instrument codes of its own class.

For the remainder of Part I we are pretending that the Europa Growth Fund has exactly one share class, so the distinction does not yet matter for narrative purposes: whenever we talk about *the fund's ISIN* in this part, what we really mean is *the ISIN of its only share class*. Part II (§5.9) will put the share-class-level identifiers in their proper place inside each `<ShareClass>`. The only identifier that actually *belongs* at the fund level is the LEI, and it will remain there throughout. Note that **FundsXML uses the name `GermanWKN` for what German market partici-**

pants usually call a WKN, and **SwissValorenCode** for what the Swiss call a **Valor**. The rest of this chapter uses the short names *WKN* and *Valor* in prose and the schema names in XML fragments; readers writing code against the schema should prefer the long forms.

One point deserves to be made now, even before we look at any single identifier in detail: the **LEI of the fund is not the LEI of its asset manager**. Chapter 4 introduced the LEI of *Europa Asset Management S.A.* (529900T8BM49AURSD055) as part of the `DataSupplier` block. The Europa Growth Fund itself, as a separate legal entity, has its own LEI — 549300ABCDEFGHJKLM34 — and it is this second identifier that appears in `Fund/Identifiers/LEI`, not the first. Confusing the two is a surprisingly common mistake among producers who have built their pipelines around a single "company LEI" and forgotten that a fund is a legal entity in its own right.

5.4.2 The Core Identifiers in Detail

We take the five most important identifier families in turn. Each entry below gives the structure of the identifier, its issuing body, the level at which FundsXML carries it, and the single most important operational rule associated with it.

ISIN (International Securities Identification Number) is a twelve-character alphanumeric code defined by ISO 6166. The first two characters are the country of issue, the next nine are a national identifier, and the twelfth is a check digit computed by the Luhn-like algorithm of the standard. ISINs are issued by national numbering agencies — in Luxembourg, the *Clearstream Banking* numbering agency — and are the primary key of cross-border fund distribution in Europe. Every separately investable share class has its own ISIN; a fund with three share classes has three ISINs. At the fund level, strictly speaking, there is no single ISIN — only the collection of its share classes' ISINs — but many producers expose the principal class's ISIN at `FundStaticData` level as a shortcut for display purposes. The schema tolerates this as long as the same ISIN also appears on its owning share class.

WKN (Wertpapierkennnummer) is a six-character alphanumeric code assigned by the German central numbering agency. It predates the ISIN by several decades and remains mandatory in German fact sheets, German regulator filings, and the displays of German retail broker systems. Every German-distributed share class has a WKN in addition to its ISIN; the two coexist and the same share class carries both. Producers who serve German consumers at any volume populate WKN for every distributed share class automatically.

Valor is a five- to nine-digit numeric code issued by SIX Financial Information in Switzerland. It is the Swiss counterpart to WKN: mandatory for distribution into Switzerland, mandatory for Swiss stock-exchange listings, and expected by every Swiss consumer of fund data. The Europa Growth Fund, which is distributed into Switzerland, carries Valor numbers on its CHF-hedged share class (and often on its EUR classes as well, since Valor can identify any security that a Swiss intermediary may hold on behalf of a client).

LEI (Legal Entity Identifier) is a twenty-character alphanumeric code defined by ISO 17442, issued by local operating units of the Global Legal Entity Identifier Foundation (GLEIF). Unlike ISIN, WKN, and Valor, the LEI identifies a *legal entity*, not a financial instrument. At the fund level, the LEI uniquely identifies the fund itself; at the umbrella level (which we will come to in §5.14), each sub-fund under the umbrella carries its own LEI and the umbrella carries another. LEIs are required for regulatory reporting under MiFID II, EMIR, AIFMD Annex IV, and SFDR, and producers who forget to populate them will find their deliveries rejected by regulator-facing consumers. Unlike the instrument identifiers, the LEI is also stable across corporate events — a rebranding, a merger, a translation of a legal name — which is why Chapter 4 recommended matching allowlists on LEI rather than on name.

CUSIP (nine-character, North-American) and **SEDOL** (seven-character, British) appear occasionally on European funds that are marketed to international investors or listed outside Europe. They are less common than the first four, and FundsXML carries them as optional fields when they apply.

Table 5.2 — Fund identifier families

Schema element	Issuer	Level	Typical use
ISIN	National numbering agencies (ISO 6166)	Share class	Cross-border distribution, trading, reference data
GermanWKN	German WM-Datenservice	Share class	German factsheets, retail broker systems, BaFin filings
SwissValoren Code	SIX Financial Information (Switzerland)	Share class	Swiss distribution, SIX Swiss Exchange listings
LEI	GLEIF local operating units	Fund / sub-fund	Regulatory reporting under MiFID II, EMIR, AIFMD, SFDR
CUSIP	CUSIP Global Services (US/Canada)	Share class	Funds marketed to North-American investors
SEDOL	London Stock Exchange (UK)	Share class	British listings and broker systems

5.4.3 OtherID — the Escape Hatch

No schema can enumerate every identifier that every consumer in every jurisdiction will ever need, and FundsXML does not try. For everything that the schema does not define explicitly — internal fund-administrator reference codes, national regulator codes, proprietary distributor identifiers, Morningstar fund IDs — there is the **OtherID** mechanism. An **OtherID** element sits inside `<Identifiers>` alongside the named children, carries a text value, and takes one of two attributes: `ListedType`, whose value is drawn from a long enumeration of recognised schemes (among them `INTERNAL FUND CODE`, `CSSF FUND CODE`, `BAFIN ID`, `AUSTRIAN FUND CODE`, `CNMV CODE`, `INAV BLOOMBERG LISTING CODE`, `REUTERS LISTING CODE`, and many others), or `FreeType`, an open string for schemes the enumeration does not cover. Consumers read the attribute to decide whether they recognise the scheme and, if so, how to interpret the value.

The escape hatch is valuable precisely because it is structured: unlike a generic "extra fields" blob, **OtherID** forces the producer to declare the scheme of every extra identifier, so that a consumer can decide field by field whether it knows what to do with each one. A typical production `<Identifiers>` block at the fund level carries the LEI and one or two **OtherID** entries — the producer's internal fund code under `ListedType="INTERNAL FUND CODE"`, perhaps the national regulator's fund code under `ListedType="CSSF FUND CODE"` for a Luxembourg fund. For producer-internal reference codes, `OtherID ListedType="INTERNAL FUND CODE"` is the idiomatic way to transport what some producers historically called a *DataSupplierFundID*; there is no dedicated element with that name in the schema.

The rule of discipline that matters is that an `OtherID` must **never** be used for an identifier that has a dedicated field. Writing the ISIN into an `<OtherID ListedType="...">ISIN</OtherID>` element instead of into the proper `<ISIN>` element is a recurring mistake — often the result of a mechanical generator that was configured before the producer learned about the dedicated element — and it breaks downstream consumers who look for ISIN at its expected location. The symptom is a consumer that "cannot find" the ISIN of a fund even though it is right there in the document; the fix is to move the identifier into its proper place and, often, to audit the producer for the same mistake on other fields.

5.5 Multilingual Names and Descriptions

Design principle 4 from Chapter 3 — *multi-language by design* — meets the practicalities of European fund distribution in this section. A single UCITS may be sold in eleven countries under the same ISIN with the same portfolio and the same investment policy, but with eleven different marketing names, eleven translations of its investment objective, and eleven versions of its risk-and-reward narrative. Producing eleven separate FundsXML documents, one per language, would defeat the point of a standard; FundsXML therefore allows every textual field to carry several language variants side by side, so that a single delivery can serve every distribution country at once.

5.5.1 The `Names` Container and the `Language` Attribute

FundsXML models fund names through a single `<Names>` container (schema type `NamesType`) whose children are **named** rather than polymorphic. A producer that wants several language variants does *not* repeat a generic `<Name>` element with different `language` attributes at the top of `<Names>`. Instead, `<Names>` holds a sequence of specifically named children — `OfficialName`, `FullName`, `MarketingName`, `ShortName`, `PreviousName`, `DynLenNames`, `LanguageNames` — and only the last of these, `LanguageNames`, is a repeatable container in which individual `<Name>` elements each carry a `language` attribute. The design keeps the single-language fast path simple (one `<OfficialName>` and done) and concentrates the multilingual machinery in one place.

A complete Europa Growth Fund `<Names>` block therefore looks like this:

```
<Names>
  <OfficialName>Europa Asset Management Investments – Europa Growth Sub-
Fund</OfficialName>
  <MarketingName>Europa Growth Fund</MarketingName>
  <ShortName>EGF</ShortName>
  <LanguageNames>
    <Name language="de">Europa Wachstumsfonds</Name>
    <Name language="fr">Europa Fonds de Croissance</Name>
    <Name language="it">Europa Fondo di Crescita</Name>
  </LanguageNames>
</Names>
```

Three things about this block are worth stating explicitly. First, the `<OfficialName>` is **mandatory** — every `<Names>` container must have exactly one, and it is the only element inside `<Names>` that is required. It carries the fund's primary canonical name in the producer's canonical language, typically the language of the governing jurisdiction. Second, `<MarketingName>` and `<ShortName>` are optional fields for the public-facing name and its compact abbreviation; when they are present they apply *across languages*, because a single marketing name and a single short code are what distributors and broker systems actually display. Third, the `language` attribute inside `<LanguageNames>` takes an ISO 639-1 code — *en*, *de*, *fr*, *it*, *es*, *nl*, and so on — and each `<Name language="xx">` element carries exactly one translation.

A consumer that needs a localised variant picks the matching `LanguageNames/Name` ; if no variant matches its preferred locale, it falls back to the `<MarketingName>` (for display purposes), to the `<OfficialName>` (for legal purposes), or to a configured default language. The fallback discipline is a consumer-side concern; the producer's job is simply to populate as many language variants as the fund's distribution agreements require, and to populate them consistently. Producers should populate *all* relevant languages in every delivery, not only the language of the intended recipient. A single FundsXML file routed to a French distributor should still carry the German and Italian names alongside the French one, because downstream the distributor may feed the same file into a factsheet engine that renders in any of those languages. Filtering to a single language on the producer side is tempting and wrong.

5.5.2 Types of Names

The children of `<Names>` come in several shades, and they mean subtly different things. Confusing them is a routine cause of factsheets that display the wrong text.

OfficialName is the fund's full legal name as it appears in the prospectus and in any court or regulatory filing. It is mandatory, and it is the only name that the producer is obliged to populate. For the Europa Growth Fund the official name is *Europa Asset Management Investments — Europa Growth Sub-Fund*, reflecting the sub-fund relationship even in the standalone-looking listings of this chapter. The official name is typically held in a single canonical language matching the governing jurisdiction; when a jurisdiction requires translations of the legal name, the translations go into `LanguageNames` alongside the marketing translations.

FullName is an optional longer form that is used in umbrella structures to disambiguate a sub-fund from its umbrella. When a producer prefers to keep `OfficialName` as just the sub-fund's own name — *Europa Growth Sub-Fund* — the fully qualified *Umbrella SICAV — Europa Growth Sub-Fund* form goes into `FullName` . The two coexist and serve different display contexts.

MarketingName is the marketing or commercial name of the fund — the name that appears on fact sheets, client portals, and advertising. For the Europa Growth Fund this is simply *Europa Growth Fund*. It is the name the producer wants end-users to see; it is typically shorter and friendlier than the legal name and may differ from it substantially.

ShortName is an abbreviation or short code, used in compact displays such as tables and trading screens. For the Europa Growth Fund it is *EGF*. The short name is *not* intended to be multilingual: short codes are typically culturally neutral, and a single string serves all languages.

PreviousName carries the most recent previous name of the fund after a rebranding, along with an `@until` attribute that names the date on which the previous name ceased to apply. Consumers that maintain historical reference-data joins read the previous name alongside the current one so that older data with the old name can still be matched

to the fund.

DynLenNames carries several length-constrained variants of the current name for display systems that truncate at fixed widths — a 16-character short form, a 32-character medium form, a 64-character long form. Producers use it when their consumers include legacy terminals that would otherwise butcher the marketing name. It is optional and rarely populated in modern deliveries.

LanguageNames is the container for translated variants, as we saw in §5.5.1. Each `<Name>` child inside it carries a two-letter ISO 639-1 `language` attribute and one translation.

Producer-internal reference codes — what some producers historically called a *DataSupplierFundID* — do **not** live inside `<Names>`. They live inside `<Identifiers>` as an `OtherID` entry with `ListedType="INTERNAL FUND CODE"`, as §5.4.3 described. This is worth saying twice because the historical name of the field misled an entire generation of integrators into looking for it in the wrong place.

5.5.3 FundTexts — Investment Objectives, Policies, and Narratives

Longer textual fields — *investment objective*, *investment policy*, *risk profile narrative*, *investor profile description* — do not live inside `<Names>`. They live in a separate container, `FundStaticData/FundTexts`, whose `<FundText>` children each carry one piece of free-form text alongside its metadata. These are the fields that feed factsheet generators and client-portal descriptions, and their quality determines whether a fund reads like a carefully prepared product or like a machine translation.

Each `<FundText>` element has the following shape. A `<Language>` child (mandatory) names the language of the content using an ISO 639 code. A `<Date>` child (mandatory) names the date on which the text is valid — important because investment-policy wording is revised periodically and consumers need to know which version they are reading. One of `<ListedType>` or `<UnlistedType>` (a mandatory choice) names the kind of text: the listed enumeration covers the most common regulated categories — `INVESTMENT STRATEGY`, `INVESTOR PROFILE`, `RISK DESCRIPTION`, `RISK MANAGEMENT`, `FEES`, `BENCHMARK`, `SUBS REDS`, `LEGAL INFORMATION`, `TAX INFORMATION`, `ANNUAL REPORT`, `SEMI ANNUAL REPORT` — and `<UnlistedType>` takes a short free-form label for text categories that the enumeration does not cover. An optional `<Title>` supplies a display title in the chosen language. A `<Content>` element (mandatory) carries the text itself. An optional `<CountriesWhereApplicable>` lists the distribution countries in which this text variant is permitted, for producers that ship jurisdiction-specific wording.

A fund that carries an investment objective in English and German therefore emits two `<FundText>` elements as siblings inside `<FundTexts>`, one per language, each with the same `<ListedType>INVESTMENT STRATEGY</ListedType>` and the same `<Date>`. A fund that additionally carries a risk description emits further `<FundText>` elements with `<ListedType>RISK DESCRIPTION</ListedType>`. A single `<FundTexts>` block in a realistic delivery typically contains between six and twenty entries, covering the main text categories in the main distribution languages.

Two rules of practice are worth following.

First, **the first sentence of `<Content>` must stand alone**. Many downstream consumers — compact fact-sheet templates, distributor search results, one-line product cards — truncate longer descriptions after a fixed number of characters, typically one hundred and fifty to two hundred. A description whose first sentence only makes sense in

the context of the second will look mangled in every such consumer. Producers should write the first sentence as a self-contained summary and use the remaining sentences for elaboration.

Second, **do not embed HTML or other markup** in `<Content>`. Some consumers escape the markup, some pass it through, and some strip it entirely, and the resulting inconsistency is a well-known source of visible bugs. If a description needs formatting, handle the formatting on the consumer side, or use the root-level `<Documents>` section (Chapter 9) to deliver a separate rich-text version as a PDF or as structured `<BinaryData>`.

One interaction is worth noting here for later reference. Chapter 8 treats the FinDatEx regulatory templates (EMT, EPT, EET), and these templates define their own normalised text fields for certain regulated disclosures — the PRIIPs risk narrative, the SFDR sustainability statement, the MiFID target-market description. Where a regulated text field exists, its regulatory version *overrides* the informal `<FundText>` entries for the purpose of regulatory consumers, but the informal version continues to serve unregulated consumers such as marketing factsheets. Producers should populate both consistently; consumers should read whichever is appropriate to their use case.

5.6 Fund Classification

Every fund in FundsXML carries a set of classification fields that describe what the fund *is* — in legal, geographic, and investment terms. The classification fields are not the same as the regulatory templates of Chapter 8; they are the fund's own self-description, used by factsheet engines, by distributor search and filter tools, and by internal reference systems. We treat them in three groups: legal form and origin, investment characteristics, and risk-and-benchmark indicators. Almost everything in this section lives inside `FundStaticData`, at the fund level; registration countries are the one exception and belong to each individual share class (§5.10).

5.6.1 Legal Form, Domicile, and Inception

The first group answers the question *where does this fund come from, and under what legal form?*

DomicileCountry is the ISO 3166-1 alpha-2 code of the jurisdiction in which the fund is legally domiciled, carried as the very first child of `FundStaticData`. For the Europa Growth Fund this is `LU` — Luxembourg, the European fund domicile of choice for cross-border UCITS. The domicile determines which supervisor regulates the fund, which legal form it can take, and which tax regime applies at the fund level.

Registration countries are not a fund-level field. A fund may be domiciled in one country and *registered for distribution* in several others; in fact, cross-border distribution is the whole point of the UCITS regime. The Europa Growth Fund is domiciled in Luxembourg and registered for distribution in eleven countries: Luxembourg itself, France, Germany, Italy, Spain, the Netherlands, Austria, Belgium, Portugal, Switzerland, and Sweden. A natural first instinct is to model this as a list of `CountryOfRegistration` children on the fund, but the schema puts the list one level lower: `ShareClass/RegistrationCountries/RegistrationCountry`, with each entry naming a country code and an operational status (`Registered` or `De-registered`). The reason is that two share classes of the same fund may be registered in different countries — the institutional class may be authorised in fewer jurisdictions than the retail class — and a single fund-level list cannot express that. §5.10 returns to this.

Legal structure is expressed through a choice between `ListedLegalStructure` (an enumerated field covering the common European vehicle types) and `UnlistedLegalStructure` (a free-text fallback). Both alternatives are declared with `minOccurs="0"`, which means the schema accepts a `FundStaticData` block that omits the legal structure entirely — producers may do so for funds whose legal nature is carried in country-specific extensions instead. In practice every realistic European fund populates one of the two, and for the Europa Growth Fund we always pick `ListedLegalStructure`. The enumeration is specific and combines the regulatory regime with the corporate form in single values: `UCITS`, `UCITS - SICAV`, `UCITS - SICAF`, `UCITS - CONTRACTUAL TYPE`, `AIF`, `AIF - HEDGE FUND`, `AIF - PRIVATE EQUITY FUND`, `AIF - VENTURE CAPITAL FUND`, `AIF - REAL ESTATE FUND`, `AIF - REIT`, `AIF - INFRASTRUCTURE FUND`, `AIF - COMMODITY FUND`, `AIF - SOVEREIGN WEALTH FUND`, `AIF - ELTIF`, `AIF - EUVECA`, `AIF - EUSEF`, and `SPV`. Because the values already combine regime and corporate form, a producer does **not** emit two separate entries for `UCITS` and `SICAV`: the Europa Growth Fund, which is a UCITS operating as a SICAV, is represented by a single `<ListedLegalStructure>UCITS - SICAV</ListedLegalStructure>`. The enumeration is the tricky part of this field — producers that try to model the regulatory regime and the corporate form as two independent fields invariably drift away from the schema, and consumers that try to parse the enumerated value into its two halves regret the effort.

InceptionDate, a direct child of `FundStaticData`, is the date on which the fund first opened for subscription. It is a single `xs:date` value, and it never changes once the fund is live. The inception date is the anchor for every *since-launch* performance figure, and consumers that compute annualised returns rely on it. One subtlety deserves an early mention: the fund-level inception date may be earlier than the inception dates of some of the fund's *share classes*, because share classes can be added to an existing fund long after the fund itself was launched. Every `<ShareClass>` element therefore also carries its own `<InceptionDate>`, and Part II (§5.10.3) will treat share-class-level inception in detail.

The schema also carries a handful of adjacent date and status fields that a full `FundStaticData` block is likely to populate: `StartOfFiscalYear` and `EndOfFiscalYear` (as a `DayMonthType` pair), `OpenClosedEnded` (`OPEN` or `CLOSED`), `ClosedType` (`HARD` or `SOFT`) for funds that have stopped accepting new subscriptions, and `MaturityDate / LiquidationDate / LiquidationReason` for funds with a planned end. These are all optional, and we do not populate them for the Europa Growth Fund, which is an open-ended UCITS with no fixed maturity and no liquidation date in sight.

5.6.2 Investment Classification via Classifications

The second group of classification fields describes *what* the fund invests in. Here the schema takes a deliberately indirect approach: instead of defining a single closed vocabulary for asset class, geography, sector, and management style — which would drift out of date every time an industry body published a new taxonomy — FundsXML lets producers declare their classifications in terms of **external classification systems**. The container is `FundStaticData/Classifications`, and each child `<Classification>` carries the source of the classification, its internal type, a language, and one or more values, optionally tagged with a level attribute for hierarchical taxonomies.

The structure of each `<Classification>` is:

- a choice between `<ListedGroup>` (a known classification provider, drawn from an enumeration that includes `EFAMA`, `MORNINGSTAR`, `LIPPER`, `BLOOMBERG`, `MIFID`, `BVI`, `VOEIG`, `ESMA`, `AMF`, `WM`, `GERMAN CBCL`, `CIC`, and `CFI`) and `<UnlistedGroup>` (a free-text fallback for classifications from bodies the enu-

meration does not cover);

- an optional `<Type>` giving the specific taxonomy within the provider (for EFAMA, for example, *EFC* for the European Fund Classification; for Morningstar, *Global Category*);
- an optional `<Language>` attributing the language of the value strings;
- one or more `<Value>` elements, each with the classification label and an optional `level` attribute for hierarchies.

The Europa Growth Fund carries two classifications in parallel — one from EFAMA and one from Morningstar:

```
<Classifications>
  <Classification>
    <ListedGroup>EFAMA</ListedGroup>
    <Type>EFC</Type>
    <Language>en</Language>
    <Value level="1">Equity</Value>
    <Value level="2">Equity Europe</Value>
  </Classification>
  <Classification>
    <ListedGroup>MORNINGSTAR</ListedGroup>
    <Type>Global Category</Type>
    <Language>en</Language>
    <Value>Europe Large-Cap Growth Equity</Value>
  </Classification>
</Classifications>
```

The advantage of this approach is that it lets each consumer read the classification it recognises. A European distributor that drives its product universe from EFAMA categories reads the `EFAMA` entry and ignores the Morningstar one; a research analyst whose watchlists are organised by Morningstar category reads the Morningstar entry and ignores the EFAMA one. A consumer that recognises neither can fall back to inspecting `OfficialName` or `Fund/Currency` as a last resort, but the `Classifications` block is what exists for the purpose. The cost is that a reader cannot simply ask *"is this an equity fund?"* with a single schema-level field: the answer depends on which classification system they read. That trade-off is deliberate, and it reflects the reality of European fund distribution, where no single classification taxonomy has ever achieved the status of the canonical one.

A handful of related flags live one level deeper, under `SingleFund/SingleFundStaticData`, because they describe the fund as a *single* (non-umbrella) vehicle: `ExchangeTradedFlag`, `FundOfFundFlag`, `SocialResponsibleFlag`, and — most importantly for classification purposes — `ManagementType`, which takes the two values `ACTIVE` and `PASSIVE`. The Europa Growth Fund is `ACTIVE`. Producers that want to express finer distinctions such as `EnhancedIndex` or `IndexTracking` do so through the `Classifications` block with an appropriate `<ListedGroup>` entry.

5.6.3 Risk Indicators and Benchmarks

The third group of classification fields quantifies risk and names the benchmark against which the fund measures itself.

SFDRProductType is a fund-level field inside `FundStaticData` that names the SFDR (Sustainable Finance Disclosure Regulation) regime under which the fund is classified. It takes integer values `0`, `6`, `8`, or `9`: `6` for a fund that qualifies under Article 6 (no sustainability characteristics), `8` for an Article 8 fund (promotes environmental or social characteristics), `9` for an Article 9 fund (has sustainable investment as its objective), and `0` for funds that do not fall under SFDR at all. The Europa Growth Fund is a conventional Article 6 product and therefore carries `<SFDRProductType>6</SFDRProductType>`. The SFDR classification is the single most consequential ESG-related field in modern European fund disclosure, and consumers that drive ESG product filters read it directly.

The PRIIPs **Summary Risk Indicator** — the 1-to-7 scale familiar from KIDs and factsheets — does *not* live at the fund level as a dedicated element in the main schema. The reason is that the SRI is properly a share-class-level figure (different share classes of the same fund can sit at different SRI values, because hedging changes risk characteristics), and the regulated version of the SRI is expressed through the FinDatEx regulatory templates that Chapter 8 treats. Where producers want to carry the SRI alongside `FundStaticData` for legacy reasons, they do so through `CustomAttributes` with a name such as `SRI` and a numeric value. Consumers that need the authoritative SRI read the EMT or the PRIIPs-KID from the `<Documents>` section.

Benchmark identification lives in `FundStaticData/Benchmarks/Benchmark`, as a repeatable element whose content follows `BenchmarkStaticDataType`. Each benchmark entry carries a `<BenchmarkID>` (an internal key used to link the static description to the dynamic values in `FundDynamicData/Benchmarks`), a mandatory `<Name>` and `<Currency>`, an optional `<Provider>` (a `CompanyType` subclass with its own `<Identifiers>` and `<Name>`), an optional `<BenchmarkType>` drawn from an enumeration (`Market Index`, `Blended Benchmark`, `Custom`, `Peer Groups` and `Universes`, and others), and an optional `<BenchmarkComponents>` block for composite benchmarks. The Europa Growth Fund carries a single benchmark entry:

```
<Benchmarks>
  <Benchmark>
    <BenchmarkID>MSCI-EUR-NR</BenchmarkID>
    <Name>MSCI Europe Net Total Return EUR</Name>
    <Currency>EUR</Currency>
    <Provider>
      <Identifiers>
        <LEI>549300YZUBM5UFHQY25</LEI>
      </Identifiers>
      <Name>MSCI Limited</Name>
    </Provider>
    <BenchmarkType>Market Index</BenchmarkType>
  </Benchmark>
</Benchmarks>
```

The `<BenchmarkID>` is not itself a public identifier — it is a producer-chosen string that a later `FundDynamicData/Benchmarks/Benchmark` entry will reuse to tie dynamic level data to the static description. The schema enforces this relationship as a `key / keyref` pair: a dynamic benchmark block whose `BenchmarkID` does not match a static one is a schema violation, which is the schema's way of preventing a category of reference errors.

A consumer that computes relative performance — alpha, tracking error, information ratio — reads the benchmark's name and currency from the static block and joins against its own market-data source. §5.13 will pick up the dynamic side, where the actual benchmark level on 31 March 2026 is transported inside `FundDynamicData/Benchmarks` as a `<Value>` per date.

One forward pointer is worth placing here. Chapter 8 treats the regulatory templates that the fund embeds in the `<RegulatoryReportings>` root element, and EMT/EPT contain their own versions of the risk indicators, the cost aggregates, and the benchmark references that we are discussing now. Where both layers exist in the same delivery, the regulatory-template version takes precedence for regulatory consumers and the fund-level version serves everyone else. Producers populate both consistently. Consumers read whichever is appropriate to their use case and never try to reconcile one against the other at the millisecond level.

5.6.4 Country-Specific Extensions

A final word on classification: FundsXML is a pan-European standard, but European fund distribution is full of national idiosyncrasies that no single taxonomy can cover gracefully. The schema handles this through `CountrySpecificData`, an extension container that appears at two levels of a delivery. Inside `FundStaticData`, a `<CountrySpecificData>` element carries national reporting extensions that apply to the fund as a whole — Austrian OeNB reporting codes, French AMF sub-classifications, German BVI internal categories, and so on. The national schemas are maintained in separate files (`FundsXML4_CountrySpecificData_xx.xsd`) that follow the main schema's release cycle. A similar mechanism exists on each individual share class for class-level national extensions. The companion root-level `<CountrySpecificData>` element, which is a sibling of `<Funds>` rather than a child, carries country-specific content that is not structurally tied to a single fund. We do not populate `CountrySpecificData` for the Europa Growth Fund in this chapter — the running example deliberately stays inside the core schema — but producers that serve regulated pipelines in any one European country will almost always end up populating the relevant national extension, and the reader who meets such a block in a real delivery should recognise it as an extension point rather than as data the core schema has somehow mis-named.

5.7 Fund-Level Costs and Key Figures

The last section of Part I treats the costs that a fund charges its investors, to the extent that they apply at the fund level rather than at the share-class level. FundsXML splits the cost story into two places. **Fund-level aggregate figures** — the retrospective per-annum ongoing charges and transaction costs that regulators require on KIDs and fact-sheets — live in `FundStaticData/OngoingCosts`. **Per-class detailed fees** — management fees and other charges that differ between share classes — live inside each `<ShareClass>` element under `Fees/Fee`. We introduce the fund-level side here; §5.10 will pick up the per-class side once the share-class structure is in place.

5.7.1 Types of Costs

European fund costs come in several flavours, and the vocabulary has accumulated enough layers over the years that it is worth naming them precisely before we model them.

Management fee is the annual percentage charged by the asset manager for managing the portfolio. It is expressed as a fraction of the fund's net assets and is typically deducted daily in arrears. For a European equity UCITS aimed at retail investors the management fee might be 1.50% per annum; for an institutional class of the same fund it might be 0.75%. Management fees almost always differ between share classes and therefore live on the class.

Administration fee and **depository fee** are the charges levied by the fund's administrator and by its custodian bank, respectively. They are often bundled under a single "administration" line in marketing material but are structurally distinct in the fund's cost ledger. They are typically uniform across share classes, and producers can either absorb them into the fund-level ongoing charges figure or carry them as separate fee entries on each class, depending on whether the underlying contracts actually differ between classes.

Performance fee, where it applies, is the share of the fund's out-performance that the asset manager retains as an incentive payment. A typical structure is *"20% of out-performance above benchmark, subject to a high watermark, crystallised annually"*. The three main parameters — rate, hurdle, and the high watermark — together define a complete performance-fee model. FundsXML handles the high watermark explicitly through `ShareClass/HighWatermark`, and the rate and basis through a fee entry whose `Type` names the performance fee structure. Producers that omit either the watermark or the fee entry leave consumers unable to reconstruct the actual charge.

TER (Total Expense Ratio) and **Ongoing Charges** are the retrospective aggregate figures that sum the annual cost of running the fund — management fee, administration, depository, audit, legal, and every other charge that the fund's investors collectively pay. The two terms are often used interchangeably; strictly speaking, *Ongoing Charges* (under UCITS KIID and PRIIPs) is slightly narrower than *TER* (the older EFAMA definition) in that it excludes performance fees, but the numeric values are usually similar. FundsXML carries the value through the `OngoingCosts/OngoingCost` structure described below, whose `<CostType>` field names which of the two variants the value represents.

Entry load and **exit load**, when they apply, are one-time charges levied when an investor buys into or sells out of the fund. They are almost always set at the share-class level — one class of the same fund may charge a 3% entry load while another charges none — and we will return to them in §5.10.

5.7.2 How Fund-Level Costs Are Modelled

The fund-level cost block lives at `FundStaticData/OngoingCosts` and contains one or more `<OngoingCost>` children. Each child has a fixed shape: a mandatory `<CostType>` drawn from a short enumeration (`Ongoing Costs`, `Ongoing Charges`, `Performance Fee`, `Transaction Costs`), a mandatory `<PublicationDate>` (the date the figure was computed and published), optional `<ValidFrom>` and `<ValidTo>` dates for the period to which the value applies, a mandatory `<Percentage>` (the value itself, expressed as a decimal percentage — 1.85 means 1.85%, *not* 0.0185), and an optional `<YearlyAmount>` for the absolute amount per 10,000 units of investment in the fund's currency. A typical block for a fund whose annual ongoing charges have just been refreshed at the start of the new fiscal year looks like this:

```
<OngoingCosts>
  <OngoingCost>
    <CostType>Ongoing Charges</CostType>
    <PublicationDate>2025-06-30</PublicationDate>
    <ValidFrom>2025-07-01</ValidFrom>
```

```

    <ValidTo>2026-06-30</ValidTo>
    <Percentage>1.85</Percentage>
  </OngoingCost>
</OngoingCosts>

```

A fund that also discloses a separate transaction-costs figure (common for PRIIPs-KID purposes) would add a second `<OngoingCost>` entry with `<CostType>Transaction Costs</CostType>` and its own percentage. The structure is deliberately flat: there is no place for hurdle rates or fee-waiver schedules at this layer, because the layer is designed to carry headline figures that feed factsheets and KIDs, not the full computational machinery behind a fee.

One observation about `OngoingCost` is worth making explicitly. The `Percentage` value is given in **human-readable percent notation**: `1.85` means 1.85 per cent per annum. This matches every other percentage in the schema (type `PercentageType`), and consumers that treat the value as a fraction (0.0185) will be off by a factor of a hundred — a spectacular error and, unfortunately, a recurring one. The rule of thumb is: if the number in `<Percentage>` looks like a reasonable management fee when you read it as a percent, it probably is; if it looks like a reasonable management fee after you divide by 100, you are about to make a mistake.

The fund-level block we have just assembled is the producer's own representation of its headline costs — the representation that feeds factsheet engines and distributor search tools. Chapter 8 treats the EMT (European MiFID Template), which defines a *different* and more detailed representation of the same charges, designed specifically for cost disclosure to retail investors under MiFID II. The EMT version is the definitive one for regulatory purposes; `OngoingCosts` in `FundStaticData` is the fund's operational view. Both exist in the same delivery, are populated consistently, and serve different consumers. The per-share-class detailed fee structure — `ShareClass/Fees/Fee` with its `ShareClassFeeType` — is the third layer, covered in §5.10, and it is where management fees and any class-specific surcharges actually live.

5.8 From One Share Class to Many

So far we have pretended that the Europa Growth Fund has exactly one share class. It does not. It has three, and they are different enough from each other that treating them as one would hide most of what makes real European fund distribution interesting. Part II relaxes the pretence and walks through the topics on which the fund-level picture branches.

The three share classes of the Europa Growth Fund are worth introducing by name, because we will use all three for the rest of the chapter and again in every subsequent chapter of Part II:

- **R-EUR-ACC** — the retail euro-denominated accumulating share class. Sold to individual investors across the eleven distribution countries, priced in EUR, and thesaurising all income into the NAV rather than paying dividends.
- **R-CHF-ACC-HEDGED** — the retail Swiss-franc-denominated currency-hedged accumulating share class. Aimed at Swiss retail investors who want exposure to European equities but do not want the EUR/CHF currency risk that would come with holding the ordinary euro class. The portfolio is the same as the R-EUR-ACC class; the difference is a currency overlay that hedges the EUR exposure into CHF.

- **I-EUR-DIST** — the institutional euro-denominated distributing share class. Aimed at pension funds, insurance companies, and other institutional investors who want predictable income distributions and who qualify for a lower fee schedule because of their minimum investment size.

All three of these share classes belong to the same fund, hold the same portfolio, follow the same investment policy, are managed by the same team, and share the same benchmark. In everything we covered in Part I — legal form, domicile, investment classification, benchmark — they are identical. What differs between them is a specific set of fields that we now need to place inside individual `<ShareClass>` elements rather than on the fund itself.

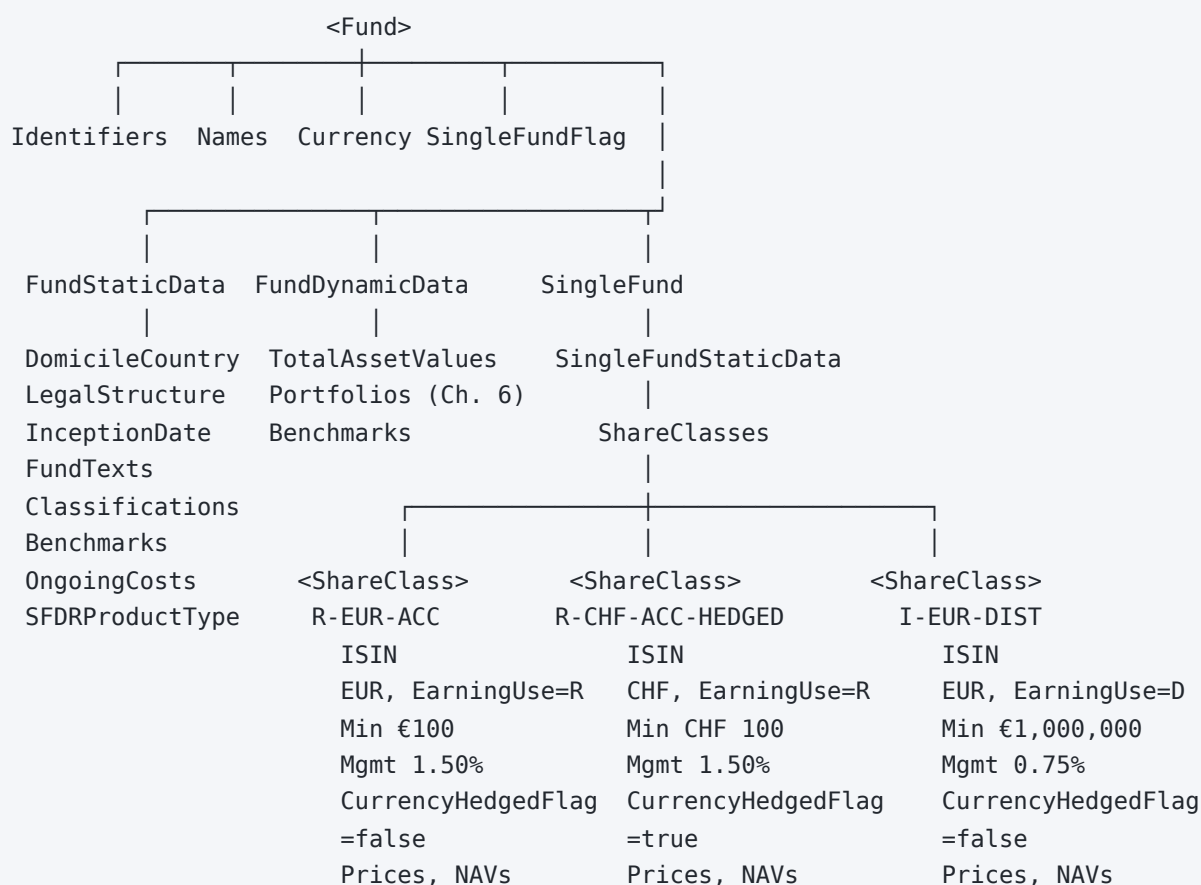
The rule that organises the placement is straightforward: **anything that differs between share classes lives on the `<ShareClass>` element; anything that is common to all classes stays on `<Fund>` and `<FundStaticData>`**. Table 5.3 shows where the main topics land when the rule is applied, using the schema names that actually appear in the XSD.

Table 5.3 — Where the main fields live once share classes are in play

Stays on <Fund> / FundStaticData	Lives inside each <ShareClass>
Fund/Identifiers/LEI (fund LEI)	ShareClass/Identifiers/ISIN , GermanWKN , SwissValorenCode
Fund/Currency (fund base currency)	ShareClass/Currency (class currency)
FundStaticData/DomicileCountry , ListedLegalStructure	ShareClass/RegistrationCountries/RegistrationCountry
FundStaticData/InceptionDate (fund inception)	ShareClass/InceptionDate (class inception)
FundStaticData/FundTexts (investment strategy, risk)	ShareClass/ShareClassType (Code , EarningUse)
FundStaticData/Classifications , Benchmarks	ShareClass/SubscriptionRestrictions/MinSubscriptionAmount
FundStaticData/OngoingCosts (headline costs)	ShareClass/Fees/Fee (per-class management fees, ...)
FundStaticData/SFDRProductType	ShareClass/CurrencyHedgedFlag
FundDynamicData/TotalAssetValues (fund aggregate)	ShareClass/Prices/Price (NAV per unit)
FundDynamicData/Portfolios (→ Chapter 6)	ShareClass/TotalAssetValues (class-level TNAV, shares outstanding)
FundDynamicData/Benchmarks (dynamic benchmark values)	ShareClass/PerformanceFigures (per-class returns)
SingleFund/SingleFundStaticData (ManagementType , flags)	ShareClass/Flows , ShareClass/Distributions (→ Chapter 7)

Two consequences of the table deserve to be named explicitly. First, **everything from Part I still applies**. Share classes do not replace the fund-level fields; they add a new layer of per-class variation. A reader who has understood Part I does not need to unlearn anything in Part II; they just need to learn which of their understood fields now reappear, under schema-correct names, inside `<ShareClass>`. Second, **the portfolio is shared across all share classes**. Regardless of how many share classes a fund has, they all ultimately own shares of the same portfolio; the portfolio is a property of the fund, not of its share classes. Chapter 6 walks through the portfolio in detail, and the reader will notice that `FundDynamicData/Portfolios` hangs off the `Fund` level rather than off `ShareClass`. The share classes differ on how *claims* against the portfolio are structured — what currency the claims are denominated in, what fees they bear, whether they pay out income — but the portfolio itself is singular. (A share class *can* carry its own `ShareClass/Portfolios` if the producer reports the portfolio at share-class level, but this is an advanced pattern that the book does not pursue.)

Figure 5.2 — Fund with three share classes



The remainder of Part II walks through the topics that the table and the figure put on the share-class side, in an order that mirrors the order of Part I. We begin with identity and currency (§5.9), because those are the topics that differ most visibly. We then treat the behavioural properties that distinguish share classes — earning use, hedging, minimum investment, inception, registration (§5.10). Finally, we look at the patterns that recur across funds and across asset managers when share classes multiply (§5.11), so that the reader leaves Part II with a catalogue of the commoner multi-share-class configurations and can recognise them in the wild.

5.9 ShareClass Identity and Currency

5.9.1 Identifiers at the Share Class Level

Each `<ShareClass>` element begins, just like `<Fund>`, with an `<Identifiers>` container. The container is the same schema type (`IdentifiersType`) as the one at fund level; it accepts the same children; and it is populated with whichever instrument-level codes name the class. ISIN, GermanWKN, SwissValorenCode, CUSIP, SEDOL, and the optional proprietary `OtherID` entries all live here.

The three share classes of the Europa Growth Fund carry three distinct ISINs:

- *R-EUR-ACC* — LU2100000011
- *R-CHF-ACC-HEDGED* — LU2100000029

- *I-EUR-DIST* — LU2100000037

Each of the three is a twelve-character ISO 6166 identifier; each is issued by the Luxembourg numbering agency (LU prefix); and each refers to exactly one share class. An investor subscribing to the retail euro class buys units whose ISIN is LU2100000011; that investor cannot, by owning units against that ISIN, gain exposure to the CHF-hedged variant without explicitly switching classes. The three ISINs are functionally three different products even though they are backed by the same portfolio.

GermanWKN, SwissValorenCode, CUSIP, and every other share-class-level identifier from §5.4.2 follow the same pattern: one entry per class inside the class's `<Identifiers>` block. A German investor who receives a fact-sheet for the Europa Growth Fund sees three WKNs — one per class — and picks the one that matches the class they want to buy. A Swiss investor sees three Valors. A producer that gets this wrong, and writes a single WKN at the fund level when it should have been three at the class level, will find that German consumers cannot tell the classes apart and routinely route orders to the wrong class.

A rare but real edge case deserves a mention. A share class may, over its lifetime, carry *more than one* ISIN — for example when two funds merge, and the surviving class inherits the historical ISIN of the disappearing one alongside its own. FundsXML allows this: the class's `<Identifiers>` can hold a primary `<ISIN>` together with one or more historical ISINs represented as `<OtherID>` entries with a descriptive `FreeType` attribute. Consumers that need historical continuity look at the full set; consumers that need the current identity look only at the `<ISIN>` field.

Listed share classes — exchange-traded UCITS with public order books — carry additional identification that does not fit neatly into the standard identifier fields. The ticker symbol and the market identifier code (MIC) of the exchange on which the class is listed are carried via `ShareClass/MarketPlaces/MarketPlace`, which holds a four-character `<MarketIdentifierCode>`, a `<NAVCurrency>`, a `<TradingCurrency>`, and optionally a `<MarketMaker>` block. Exchange-traded UCITS are described through this block rather than through `OtherID`, and Chapter 13 returns to the full convention for listed-class metadata.

5.9.2 Currency of the Share Class

Currency is the second field that lives on the class as well as on the fund, and the distinction between *class currency* and *fund base currency* is one of the most important conceptual points of this chapter.

The **fund base currency** is the currency in which the fund's portfolio is primarily denominated and in which the fund's own books are kept. For the Europa Growth Fund this is EUR: the portfolio consists almost entirely of European equities priced in euros, the custody accounts are maintained in euros, and the fund-level total net assets figure is computed in euros. The base currency is a property of the fund, carried in `Fund/Currency` as a direct child of `<Fund>` (not inside `FundStaticData`), and it is one of the four mandatory fund-level elements we saw in §5.2.

The **share-class currency** is the currency in which the NAV of an individual share class is expressed and in which investors subscribe and redeem against that class. It lives in `ShareClass/Currency`, the direct counterpart of `Fund/Currency`, and it may or may not coincide with the base currency. For the Europa Growth Fund's R-EUR-ACC and I-EUR-DIST classes, the class currency is EUR, the same as the base currency; for the R-CHF-ACC-

HEDGED class, the class currency is CHF, different from the base currency. All three classes ultimately hold claims against the same EUR-denominated portfolio, but the claims are denominated in different currencies and the claim-holders see different NAV numbers.

The mechanism that makes a CHF class possible when the underlying assets are in EUR is **currency hedging**. The fund enters into a rolling programme of currency forwards (or another hedging instrument) that converts the EUR exposure into CHF exposure for the purposes of the hedged class. The hedge costs money — interest-rate differentials, forward points, rolling frictions — and the cost is borne by the hedged class alone, which is why its NAV evolves differently from the ordinary euro class even though both classes hold the same underlying portfolio. §5.10 treats hedging as a behavioural property of the class in more detail.

A point of frequent confusion is worth stating plainly: **the base currency and the class currency are not interchangeable**. A consumer that reads a CHF NAV from an R-CHF-ACC-HEDGED class and treats it as a EUR number will get an answer that is wrong by the EUR/CHF exchange rate — several percent — and that error will propagate through every downstream calculation. The class element always names its own currency explicitly in `ShareClass/Currency`, and every `<Amount>` carrying a value in the class's currency has a mandatory `ccy` attribute that a consumer is expected to read and respect.

5.10 Earning Use, Hedging, Subscription, Inception, Registration, Fees

5.10.1 Earning Use: Accumulating and Distributing

The first of the behavioural properties that distinguish share classes is what happens to the income the fund's portfolio generates — dividends on equities, coupons on bonds, interest on cash. The choice is binary: the class either retains the income and reflects it in a higher NAV, or it pays the income out to investors at regular intervals.

Accumulating classes retain income. Every dividend received on the portfolio is added to the fund's net assets; since the share-class's NAV is computed as net assets divided by shares outstanding, and since the number of shares does not change when income is received, the NAV of an accumulating class rises by an amount reflecting the retained income. The investor benefits through capital appreciation of their units rather than through cash in hand. The R-EUR-ACC and R-CHF-ACC-HEDGED classes of the Europa Growth Fund are both accumulating.

Distributing classes pay income out. On a schedule defined in the prospectus — annually, semi-annually, quarterly, or in some cases monthly — the fund declares a dividend, the investors of the distributing class receive a cash payment equal to their pro-rata share of the accumulated income, and the NAV of the class drops on the ex-dividend date by the amount of the distribution. A consumer that renders the NAV of a distributing class over time sees a sawtooth pattern: smooth growth between distributions, a discrete drop on each ex-date. Both the growth and the drop are exactly as they should be — they reflect the same economic reality as the accumulating class's smoother curve, just expressed differently. The I-EUR-DIST class of the Europa Growth Fund is distributing, with an annual ex-date in mid-May each year.

FundsXML models the choice inside `ShareClass/ShareClassType`, a small structured element (schema type `ShareClassTypeType`) whose mandatory first child is `<Code>` — the producer's own class code (e.g. R, I, X, Z) — followed by three optional children: an `<EarningUse>` enumeration taking one of three single-charac-

ter values (**D** for *Distributing*, **P** for *Partly distributing* — the rare hybrid case where some income streams are paid out and others reinvested — and **R** for *Reinvesting*, the schema's word for *Accumulating*), a short `<Name>` giving a human-readable label, and a longer `<Description>` text. The Europa Growth Fund's accumulating classes therefore carry `<EarningUse>R</EarningUse>`, and the distributing institutional class carries `<EarningUse>D</EarningUse>`. `<EarningUse>` is schema-optional, so a class whose accumulating-versus-distributing nature is communicated through other fields (e.g. a custom attribute, or implicit from the class code) can omit it; consumers should therefore treat a missing `<EarningUse>` as "unspecified" rather than as a default. Note the slight terminological mismatch: what the industry calls *accumulating* is what the schema calls *reinvesting* (**R**), because the income is reinvested into the fund rather than paid out. Producers that get the two letters the wrong way round create the single most damaging classification error in the chapter.

The actual dividend events for distributing classes are emitted as `Distribution` entries inside `ShareClass/Distributions`. Each `<Distribution>` has a mandatory `<ActionCode>` (**C** / **M** / **D** for create/modify/delete), `<DividendStatus>` (**OFFICIAL** or **ESTIMATED**), optional `<AnnouncementDate>` and `<RecordDate>`, mandatory `<ExDate>`, `<PaymentDate>`, and `<PaymentCurrency>`, and mandatory `<GrossDividendAmount>` and `<NetDividendAmount>` blocks each containing a per-share amount. Chapter 7 treats the flow side in full; the point here is simply that the dividend is *not* encoded on `ShareClassType` — the class is flagged **D**, but the actual events live in a separate time-series.

The consequence for performance comparison is worth naming because it is a routine source of confusion. Two share classes of the same fund, one accumulating and one distributing, have different NAV curves but the same *total return* — the cumulative performance including reinvested distributions. A consumer that compares raw NAV series between an accumulating and a distributing class will see a divergence that is purely an artefact of distribution treatment; a consumer that compares total returns will see, correctly, that the two classes are economically similar. Chapter 10 treats this as one of the validation rules that a performance-reporting pipeline should enforce.

5.10.2 Currency Hedging

Hedging is the second behavioural property that distinguishes share classes. We described the mechanism briefly in §5.9.2; here we look at how it is represented.

The primary hedging flag is `CurrencyHedgedFlag`, a Boolean that sits directly inside `<ShareClass>`. It is *true* if the class is currency-hedged, *false* otherwise. The Europa Growth Fund's R-CHF-ACC-HEDGED class has `<CurrencyHedgedFlag>true</CurrencyHedgedFlag>`, and the other two classes have `<CurrencyHedgedFlag>false</CurrencyHedgedFlag>` (or simply omit the element). There is no separate `HedgedCurrency` element: the currency into which the class is hedged *is* the class currency itself, carried in `ShareClass/Currency`, so a hedged CHF class naturally has `<Currency>CHF</Currency>`.

A producer that wants to describe the hedging approach in more detail carries a `<FundHedgingStrategy>` element either at the fund level (as a child of `FundStaticData`) or on the individual share class. The element wraps a single `<HedgingStrategy>` enumeration whose values are `No hedge`, `Full NAV hedge`, `Full Portfolio hedge`, and `Partial hedge`. A typical retail CHF-hedged share class carries `<HedgingStrategy>Full NAV hedge</HedgingStrategy>`, meaning that the hedge is calibrated against the class's total net asset value rather than looking through to each underlying currency exposure separately. Full Portfolio hedges are rarer and appear mainly in institutional classes that want their hedge accurate at the instrument level.

One point that routinely surprises newcomers is that the hedge P&L — the profit or loss the class realises from its currency forwards — does not appear as a separate field in FundsXML. It is absorbed silently into the NAV of the hedged class, which is exactly where a disciplined investor expects it to show up: the whole purpose of the hedge is to modify the return pattern of the class, and the NAV is what captures that return. A consumer that wants to decompose the NAV into "portfolio return" and "hedge P&L" separately cannot do so from the FundsXML file alone; the decomposition is a producer-side analytic that is out of scope for the standard.

5.10.3 Subscription Restrictions, Inception, and Registration

The remaining behavioural fields concern the terms under which an investor can buy into the class, the history of the class itself, and the jurisdictions in which the class is authorised for distribution.

Subscription restrictions live in a dedicated `<SubscriptionRestrictions>` container inside each `<ShareClass>`. Its main children are:

- `MinSubscriptionAmount` — the smallest amount a new investor can subscribe to the class in their first transaction, carried as an `<Amount ccy="EUR">100</Amount>` (or the class's own currency). Retail classes typically set this at a small number — 100 EUR or 1,000 EUR — so that any individual investor can participate. Institutional classes set it much higher — 500,000 EUR, 1,000,000 EUR, 10,000,000 EUR — so that the class is effectively closed to retail participation and can justify its lower fee schedule.
- `MinSubscriptionShares` — the same restriction expressed in units of the class rather than in currency, for producers that prefer share-count thresholds.
- `MinSubscriptionAmountSubsequent` and `MinSubscriptionSharesSubsequent` — the equivalent thresholds for existing investors topping up their positions. They are typically smaller than the initial ones and are rarely zero.
- `MinRegularInvestmentAmount` and `MinRegularInvestmentShares` — the thresholds for savings-plan contributions, where a monthly direct-debit subscription has its own minimum.
- `OtherSubscriptionRestrictions` — a free-text fallback for anything the structured fields do not cover.

For the three share classes of the Europa Growth Fund the thresholds are:

- *R-EUR-ACC*: `MinSubscriptionAmount` 100 EUR, `MinSubscriptionAmountSubsequent` 50 EUR
- *R-CHF-ACC-HEDGED*: `MinSubscriptionAmount` 100 CHF (no subsequent minimum declared)
- *I-EUR-DIST*: `MinSubscriptionAmount` 1,000,000 EUR (no subsequent minimum declared)

A symmetric `<RedemptionRestrictions>` container carries the same kind of structure for the sell side — `MaxRedemptionAmount`, `MaxRedemptionShares`, `MinHoldingPeriod` (in months), and `OtherRedemptionRestrictions` — for classes that apply lock-ups or exit caps.

Investor-type restrictions are not a single flag. The schema carries them through two Booleans on `<ShareClass>` — `<RetailInvestorsOnly>` (true for classes that may not be sold to institutional investors, a rarity) and `<UK_RDR_Flag>` (true for classes that comply with the UK Retail Distribution Review, which bans trailing commissions on retail sales) — and, indirectly, through the minimum-subscription thresholds above. The *institutional* nature of the Europa Growth Fund's I-EUR-DIST class is communicated through its minimum subscription of 1,000,000 EUR rather than through a dedicated `IsInstitutional` flag — there is no such field in the schema. Producers that want to make the institutional character explicit can do so by populating `ShareClassType/Code`

with a suggestive value (I) and `ShareClassType/Name` with a descriptive label (*Institutional Distributing*). Consumers should never infer investor eligibility purely from a class code — that would re-introduce the name-parsing mistake that §5.11 warns against.

Inception date at the class level lives in `ShareClass/InceptionDate` and is the date on which the class itself first opened for subscription. It may or may not coincide with the fund-level inception date from §5.6.1. The Europa Growth Fund was launched in 2012; the R-EUR-ACC class has been live since launch and carries `<InceptionDate>2012-01-15</InceptionDate>`, identical to the fund. The R-CHF-ACC-HEDGED class was added in 2018 to capture Swiss retail demand and carries `<InceptionDate>2018-06-01</InceptionDate>`. The I-EUR-DIST class was added in 2020 after an institutional anchor investor was secured and carries `<InceptionDate>2020-09-15</InceptionDate>`.

The distinction between fund inception and class inception matters for performance reporting. The fund's *since-launch* performance refers to the inception date of the fund and makes sense only for classes that have been live since then. For classes added later, performance from dates prior to class inception does not exist and should not be shown: a *since-launch* figure for the R-CHF-ACC-HEDGED class starts in June 2018, not in 2012. Consumers that render historical performance are expected to read the class inception date and honour it; producers that emit pre-inception performance are in error, and Chapter 10 will list this among the business-validation rules.

Registration **countries** at the class level live inside `ShareClass/RegistrationCountries/RegistrationCountry`. Each entry carries a mandatory `<CountryCode>` (an ISO 3166-1 alpha-2 code), an optional `<Status>` (Registered or De-registered), an optional `<StatusDate>`, an optional `<SubStatus>` for investor restrictions specific to that jurisdiction (e.g. *Institutional Investors only*), and an optional `<MarketingDistributionStatus>` that tracks whether active marketing is currently under way in the country. The retail R-EUR-ACC class of the Europa Growth Fund is registered in all eleven distribution countries; the I-EUR-DIST institutional class is registered in a smaller subset, because institutional distribution is often limited to the jurisdictions in which the producer has an institutional sales team on the ground. Modelling registration per share class rather than per fund is what allows two classes of the same fund to be available in different sets of countries.

5.10.4 Per-Class Fees

The last behavioural topic is the class-specific fee schedule. We introduced the general cost vocabulary in §5.7; here we see how the per-class variant is structured.

Fees that differ between classes live in `ShareClass/Fees/Fee`, with each `<Fee>` following the schema type `ShareClassFeeType`. The mandatory children of a `<Fee>` are:

- `<Type>` — a free-text label for the fee, such as *Management Fee*, *Distribution Fee*, *Entry Load*, *Performance Fee*;
- `<PayReceive>` — a single character, P for *paid by the fund* (the usual case) or R for *received by the fund* (rare);

optionally followed by `<Minimum>` and `<Maximum>` as percentages of total net assets, a `<CalculationMethod>` free-text description, and a `<DataByPeriods>` container that carries the numeric values per reporting period. Each `<DataByPeriod>` inside `<DataByPeriods>` names the period with a `<BeginDate>` and `<EndDate>`, optionally a `<CurrentDate>` for ongoing periods, and a mandatory

`<Values>` block whose key field is `<FeeAsPercentageOfTNA>` — the percentage of total net assets that the fee represents over the named period. The result is a fee structure that can carry not only the headline rate but also the actual realised rate for every period the producer wants to publish.

A typical management-fee entry on the R-EUR-ACC class of the Europa Growth Fund looks like this:

```
<Fees>
  <Fee>
    <Type>Management Fee</Type>
    <PayReceive>P</PayReceive>
    <Maximum>1.50</Maximum>
    <CalculationMethod>Per annum of average net assets, accrued
daily</CalculationMethod>
    <DataByPeriods>
      <DataByPeriod>
        <BeginDate>2025-07-01</BeginDate>
        <EndDate>2026-06-30</EndDate>
        <CurrentDate>2026-03-31</CurrentDate>
        <Values>
          <FeeAsPercentageOfTNA>1.50</FeeAsPercentageOfTNA>
        </Values>
      </DataByPeriod>
    </DataByPeriods>
  </Fee>
</Fees>
```

The same structure applies to every per-class fee: a distribution fee, an entry load (where `<CalculationMethod>` names it as a one-off rather than an annual rate), a performance fee (where `<CalculationMethod>` would describe the outperformance formula and the crystallisation schedule). For performance fees, the **high watermark** is carried separately in `ShareClass/HighWatermark`, whose fields are `<ExistingFlag>` (Boolean, true if the class has a watermark), `<Percentage>`, `<Amount>` (the reference NAV), `<Description>` (free text), `<PeriodicalResetFlag>`, and `<ResetFrequency>`. A full performance-fee specification therefore spans a `<Fee>` entry and a `<HighWatermark>` entry, each carrying the part of the story that belongs to it.

One point of confusion that often catches newcomers is the number format of `<FeeAsPercentageOfTNA>` and `<Maximum>`. Both use `PercentageType`, which is human-readable per-cent: 1.50 means 1.50%, not 0.0150. The same rule applies as for `OngoingCost/Percentage` in §5.7.2: if the number looks like a reasonable fee when read as a percent, it is; if it looks reasonable only after division by a hundred, the producer has emitted the value in the wrong format.

5.11 Multi-Share-Class Patterns in Practice

We close Part II with a short catalogue of the patterns that recur in real European fund structures when share classes multiply. The patterns are not defined by FundsXML; they are industry conventions that producers follow and that consumers should recognise. Knowing the patterns makes reading an unfamiliar fund's share-class structure much

faster.

5.11.1 Share Class Families and Naming Conventions

The fund industry has evolved a rough alphabet of single-letter codes that mark families of share classes. The conventions vary slightly between asset managers but the common letters are stable enough to be a useful first filter.

- **A** — retail, often with an entry load of 3–5%
- **B** — retail, often distributing, and typically carrying the same fee structure as A
- **C** — retail, with no entry load but a higher management or ongoing-charges figure to compensate
- **D** — retail, specialised variants (distributing, hedged, clean-priced)
- **I** — institutional, lower fees, higher minimum investment
- **X** — super-institutional or omnibus, fees negotiated on a segregated mandate basis
- **Z** — zero-retrocession, for fee-based advisers who bill their clients directly
- **R** — retail, modern replacement for **A** in some post-MiFID-II structures with no entry loads

These letters appear in the fund's marketing material, in factsheets, and in the `OfficialName` or `ShortName` fields of the `<ShareClass>/Names` block, often combined with currency, earning use, and hedging flags: "R-EUR-ACC", "I-USD-DIST-HEDGED", "A-CHF-ACC". The Europa Growth Fund's three classes fit the pattern: *R* for retail, *I* for institutional, currencies and earning-use policies spelled out.

One rule is worth stating loudly: **the letters are not schema-enforced enumerations, and consumers must not parse them**. A class called *R-EUR-ACC* is retail, euro-denominated, and accumulating not because of its name but because of its `Currency`, its `ShareClassType/EarningUse`, its `CurrencyHedgedFlag`, and its `SubscriptionRestrictions/MinSubscriptionAmount` fields. A consumer that tries to extract those properties from the name — by string-matching on "R", "EUR", and "ACC" — will eventually meet an asset manager whose naming convention uses different letters, or who combines letters in an order the consumer did not expect, or who simply does not follow a pattern at all. The schema fields exist precisely so that consumers can read the properties authoritatively. Producers populate the names for human eyes; producers populate the fields for machines.

5.11.2 Currency-Hedged Variants

The most common multi-class pattern in cross-border European distribution is a base class paired with one or more currency-hedged variants. The base class holds the fund's native currency (typically EUR for a European fund, USD for a global fund); the hedged variants cover the other major currencies in which the fund is distributed (CHF, GBP, JPY, occasionally SEK or NOK).

Structurally, the base class and its hedged siblings share everything that is a property of the fund — portfolio, investment policy, benchmark, fee schedule, team — and differ only on:

- `Identifiers` (ISIN, GermanWKN, SwissValorenCode: each variant has its own)
- `Currency` (EUR on the base, CHF/GBP/JPY/... on each hedged variant)
- `CurrencyHedgedFlag` (`false` on the base, `true` on each hedged variant)
- `FundHedgingStrategy` (typically `No hedge` on the base, `Full NAV hedge` on each hedged variant)

- `InceptionDate` , if the hedged variants were added later than the base
- NAV and per-class `TotalAssetValues` , which diverge from the base immediately after inception because of the hedge P&L

Note what does *not* differ: the per-class fee schedule is typically identical between the base class and its hedged variants, because the hedge cost is absorbed into the NAV rather than charged as a fee. A consumer that wants to identify the hedged siblings of a given class has no direct schema-level mechanism to do so — FundsXML does not define an `IsHedgedVariantOf` pointer — and has to rely on naming conventions or on producer-supplied metadata in `CustomAttributes` . Chapter 13 recommends a project-level convention for producers that have several hedged siblings per base class.

5.11.3 Accumulating and Distributing Twins

The second common pattern is a pair of twin classes that share everything except distribution policy: one accumulates income, the other distributes. The pattern is driven by tax treatment — in many jurisdictions, accumulation is more tax-efficient for long-term savers while distribution is preferred by retirees and institutions that need regular cash flow — and by the typical size of the two investor bases.

The twins share portfolio, fee schedule, and currency. They differ on:

- `Identifiers` (each has its own ISIN, GermanWKN, and so on)
- `ShareClassType/EarningUse` (`R` on the accumulating twin, `D` on the distributing twin)
- `Distributions` — populated on the distributing twin with the actual dividend events, empty on the accumulating one
- Often `InceptionDate` (one twin is typically added later than the other)
- NAV and per-class `TotalAssetValues` , which diverge on the first ex-dividend date and remain divergent thereafter even though total returns are the same

A common trap is to compare the NAVs of twin classes directly over time and conclude that one is outperforming the other. This is almost always an artefact of distribution treatment, not of actual economic divergence. The correct comparison is total return with reinvested distributions, and Chapter 13 will treat this as one of the validation rules that a performance-reporting pipeline should enforce.

5.11.4 When Share Classes Diverge More Deeply

The patterns we have discussed so far — currency variants, accumulating/distributing twins, retail/institutional splits — account for the great majority of real share-class structures. A minority of funds carry classes that differ more deeply: different fee structures, substantially different minimum investments, occasionally even different fee-distribution mechanisms (for example, an adviser class whose distribution commission is built into the NAV versus a clean class whose distribution commission is billed separately to clients). These configurations are all legal and all representable in FundsXML.

What is *not* legal, and not representable, is a configuration in which two share classes ultimately own different portfolios. The moment two claim groups diverge at the portfolio level, they are no longer two share classes of the same fund — they are two funds, each with its own portfolio. In legal terms this is a sub-fund structure, and it belongs under the umbrella construct that §5.14 treats. Producers occasionally try to model sub-funds as share classes for

operational reasons; the result is a FundsXML document that appears internally inconsistent (two share classes, but reconciliation with the shared portfolio impossible) and that fails business validation at the consumer. The rule is straightforward: if the classes share a portfolio, they are share classes; if they do not, they are separate funds or separate sub-funds.

A forward pointer is worth placing here. Chapter 8 treats the EMT regulatory template, and EMT flattens the share-class structure further than FundsXML does: for MiFID cost disclosure purposes, each share class is represented as a separate "product" with its own cost structure, and the underlying fund-level relationships are mostly implicit. The two representations — FundsXML's hierarchical view and EMT's flattened view — serve different consumers and should be populated consistently. Chapter 8 explains how.

5.12 Dynamic Data — NAVs, Prices, and Total Net Assets

We now turn to the dynamic half of the `<Fund>` element. Up to this point we have treated it as "the fields that change every valuation day", which is a correct first approximation, and we have listed the topics it contains. With the share-class structure now in place, we can populate it with meaningful numbers — and, more importantly, we can say clearly where in the XML each figure lives, because the naive expectation that *everything dynamic lives inside FundDynamicData* is not quite right.

5.12.1 Where Dynamic Data Lives, and What Goes Where

Dynamic fund data is split between **three** places in the schema.

- **FundDynamicData** (sibling of `FundStaticData` under `<Fund>`) carries only three substructures: `TotalAssetValues` for **fund-level aggregate** totals, `Portfolios` for the shared portfolio composition, and `Benchmarks` for dynamic benchmark level values. It is deliberately small.
- **ShareClass** (each class inside `SingleFund/ShareClasses`) carries almost everything that varies per class: `Prices` (NAV per unit), `TotalAssetValues` (per-class aggregate), `Distributions` (dividend events), `Flows` (subscriptions and redemptions), `PerformanceFigures` (returns), and `Portfolios` if the producer reports the portfolio at share-class rather than fund level.
- **SingleFund/SingleFundStaticData** carries a handful of flags and classifications that are technically static but describe the single-fund identity (management type, exchange-traded flag, price-published flag).

This chapter treats the dynamic values that matter most to end-users: per-class NAVs (§5.12.2), fund-level totals (§5.12.3), and performance (§5.13). The two substructures we leave alone, *Portfolios* and *Flows / Distributions*, are big enough topics to justify chapters of their own: Chapter 6 for portfolio, Chapter 7 for flows and distributions.

5.12.2 NAV and Total Net Assets per Share Class

The net asset value per unit — the NAV that an investor sees in their statement — is the central dynamic figure of any fund. For a fund with multiple share classes, there is no single NAV: there is one NAV per class, and the schema therefore places it on the individual class rather than on the fund as a whole. Each `<ShareClass>` carries two related substructures.

`ShareClass/Prices/Price` holds one or more `<Price>` entries, each of type `ShareClassPriceType`, carrying the NAV per unit and adjacent price quantities. The mandatory children are `<ActionCode>` (`C / M / D`), `<NavDate>`, `<PriceCurrency>` (the ISO 4217 code for the class currency), `<PriceNature>` (`OFFICIAL`, `ESTIMATED`, or `TECHNICAL`), and `<NavPrice>` (the net asset value per unit, as a decimal with up to 15 total digits and 8 fractional digits). Optional children include `<SubscriptionPrice>` and `<RedemptionPrice>` — used when subscription and redemption happen at prices that differ from the mid-NAV, for example because of an entry load or a swing-pricing factor — and `<OtherPrices>` and `<SplitFactor>` for less common cases.

`ShareClass/TotalAssetValues/TotalAssetValue` holds the per-class aggregate: the class's total net asset value and its shares outstanding. The mandatory children are `<NavDate>` (the valuation date) and `<TotalAssetNature>` (again `OFFICIAL / ESTIMATED / TECHNICAL`). The optional but routinely populated children are `<TotalNetAssetValue>` (a `FundAmountType`, i.e. a currency-tagged decimal), `<TotalGrossAssetValue>` (for funds that distinguish the two), `<SharesOutstanding>` (the number of units of the class outstanding), and `<Ratio>` (the share of the fund's total net assets that this class represents, as a percentage). A producer that wants to carry alternative valuations — a swing-priced figure, a market-priced figure, a hold-to-maturity figure — adds them in `<OtherTotalAssetValues>`.

Precision and rounding follow industry convention. NAVs are typically reported to four decimal places (occasionally to eight, as the schema allows); shares outstanding are reported as integers (occasionally to three decimal places for funds that issue fractional shares); total values are reported to two decimal places. A consumer that re-computes `NavPrice × SharesOutstanding` to verify the producer's total will often find a discrepancy of a cent or two, which is a rounding artefact rather than an error. Chapter 10 treats the tolerance rules that validation pipelines should apply.

The three share classes of the Europa Growth Fund, at the close of 31 March 2026, have the following values:

- **R-EUR-ACC:** NAV 124.5078 EUR, 1,234,567 shares outstanding, TNAV 153,712,654.83 EUR (ratio 33.09%)
- **R-CHF-ACC-HEDEDGED:** NAV 119.2011 CHF, 456,789 shares outstanding, TNAV 54,438,456.22 CHF (ratio 12.21%)
- **I-EUR-DIST:** NAV 108.3344 EUR, 2,345,678 shares outstanding, TNAV 254,115,322.56 EUR (ratio 54.70%)

In XML form, the NAV and total-asset-value blocks for the R-EUR-ACC class look like this (the other two classes are structurally identical, with their own currencies and amounts):

```
<ShareClass>
  <!-- Identifiers, Names, Currency, ShareClassType, InceptionDate, ... -->
  <Prices>
    <Price>
      <ActionCode>C</ActionCode>
      <NavDate>2026-03-31</NavDate>
      <PriceCurrency>EUR</PriceCurrency>
      <PriceNature>OFFICIAL</PriceNature>
      <NavPrice>124.50780000</NavPrice>
    </Price>
  </Prices>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
```

```

    <TotalAssetNature>OFFICIAL</TotalAssetNature>
    <TotalNetAssetValue>
      <Amount ccy="EUR" isShareClassCcy="true">153712654.83</Amount>
    </TotalNetAssetValue>
    <SharesOutstanding>1234567</SharesOutstanding>
    <Ratio>33.09</Ratio>
  </TotalAssetValue>
</TotalAssetValues>
<!-- Fees, Distributions, Flows, PerformanceFigures, ... -->
</ShareClass>

```

Three observations deserve to be made about the block. First, **the link to the share class is structural, not nominal**. The NAV and the total-asset-value entry are inside the share class's own element; there is no `<ShareClassISIN>` link of the kind a newcomer might expect, because the ISIN is already the class's own identifier at the top of the class, and a consumer reading the file walks the tree rather than matching by string.

Second, **every `<Amount>` element carries a mandatory `ccy` attribute** that names the currency of the value. The R-CHF-ACC-HEDGED class's NAV is in CHF, not EUR, and the attribute makes this unambiguous — any consumer that ignores it will produce wrong numbers, and Chapter 10 will treat the discipline of always checking currencies as a core validation rule. The `isShareClassCcy` attribute, a Boolean, is optional but useful: a producer that marks the class-currency amounts with `isShareClassCcy="true"` tells the consumer explicitly that this value is in the class's own currency, which removes any ambiguity when the same structure is reused in mixed-currency contexts.

Third, **the block is self-contained per class**. Given only the NAV, the shares outstanding, and the class currency, a consumer can compute every investor-facing value a factsheet needs for that class. The fund-level figures that come next in §5.12.3 are aggregates, not substitutes.

Delivering a daily price history

The examples so far have shown a single `<Price>` entry for a single valuation date, which is the normal case for a monthly delivery. But the `<Prices>` container can hold *multiple* `<Price>` children, each with its own `<NavDate>` — and this is exactly what happens when a producer delivers daily NAV history. A transfer agent that sends the complete March 2026 price series for the R-EUR-ACC class, for instance, would populate the container with one entry per business day. The following excerpt shows the first five trading days of the month:

```

<Prices>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-02</NavDate>
    <PriceCurrency>EUR</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>123.84210000</NavPrice>
  </Price>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-03</NavDate>
    <PriceCurrency>EUR</PriceCurrency>

```

```

    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>123.96550000</NavPrice>
  </Price>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-04</NavDate>
    <PriceCurrency>EUR</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>124.11030000</NavPrice>
  </Price>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-05</NavDate>
    <PriceCurrency>EUR</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>123.72480000</NavPrice>
  </Price>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-06</NavDate>
    <PriceCurrency>EUR</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>124.00170000</NavPrice>
  </Price>
  <!-- ... 17 further business days through 2026-03-31 ... -->
</Prices>

```

Two points are worth noting. First, the order of the `<Price>` entries within the container is not prescribed by the schema — a consumer must use `<NavDate>` to sort the series, not the document order. In practice most producers emit entries in chronological order, but a robust consumer does not rely on this. Second, the `<PriceCurrency>` and `<PriceNature>` are repeated in every entry, even though they are the same for all days; this is deliberate, because the schema allows a single `<Prices>` container to carry both an official and an estimated price for the same date, and the currency could in theory change after a redenomination. The repetition keeps each `<Price>` element self-describing.

5.12.3 Total Net Assets at the Fund Level

Beyond the per-class totals, the **fund-level** aggregate — the sum of all share-class net assets, expressed in the fund's base currency — lives at `FundDynamicData/TotalAssetValues/TotalAssetValue`, which reuses the same `TotalAssetValueType` we just saw but at the enclosing fund level. The per-class figures for the Europa Growth Fund on 31 March 2026 aggregate as follows:

- R-EUR-ACC: 153,712,654.83 EUR
- R-CHF-ACC-HEDGED: 54,438,456.22 CHF, converted at 1 CHF = 1.0420 EUR (the reference rate used by the fund administrator at the valuation point): 56,724,871.39 EUR
- I-EUR-DIST: 254,115,322.56 EUR
- **Total:** 464,552,848.78 EUR

In XML form, the fund-level aggregate block looks like this:

```
<FundDynamicData>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
      <TotalAssetNature>OFFICIAL</TotalAssetNature>
      <TotalNetAssetValue>
        <Amount ccy="EUR" isFundCcy="true">464552848.78</Amount>
      </TotalNetAssetValue>
    </TotalAssetValue>
  </TotalAssetValues>
  <!-- Portfolios (→ Chapter 6), Benchmarks (dynamic values) -->
</FundDynamicData>
```

The structure is the same one we met at share-class level: the mandatory `<NavDate>` comes first, then the mandatory `<TotalAssetNature>`, then the optional `<TotalNetAssetValue>` with its currency-tagged amount. At fund level the amount carries the `isFundCcy="true"` attribute to declare it as being in the fund's base currency, which is `Fund/Currency` (EUR, in our case).

Two points need to be stated clearly. First, **the aggregate must respect currency**. The CHF total from the R-CHF-ACC-HEDGED class cannot be added directly to the EUR totals from the other two classes; it must first be converted at a reference rate, and the rate used must be consistent with whatever the fund administrator uses internally for the same purpose. A consumer that adds the raw numbers without conversion gets a result that is wrong by several percent and meaningless. Producers should never emit an aggregate that required an implicit currency conversion without being prepared to defend the rate used; consumers should, when reconciling, compute the aggregate from the per-class values themselves to verify.

Second, **the fund-level total is the figure that regulatory thresholds hang on**. AIFMD size limits, MiFID II cost-disclosure thresholds, national marketing thresholds, and internal concentration limits all use the fund-level `TotalNetAssetValue` rather than the per-class values. A consumer that implements threshold logic reads the aggregate directly and does not try to re-derive it; if the producer's aggregate disagrees with a reconciliation computation, the discrepancy is a data-quality issue to be raised with the producer, not a licence to pick one figure over the other silently.

5.13 Performance and Other Dynamic Figures

5.13.1 Performance Figures per Share Class

Performance — the total return that an investor in a share class has earned over a given period — is the most frequently consulted field in fund data and the one most prone to being reported in inconsistent ways. FundsXML carries performance at the **share-class level**, inside each `<ShareClass>` element under `ShareClass/PerformanceFigures`, with one entry per period per class. The container begins with a mandatory `<ActionCode>` (C / M / D), then carries three optional sibling lists — `<TotalReturns>`, `<Performances>`,

and `<Ratings>` — each of which carries its own flavour of figure. Producers populate whichever combination their downstream consumers expect; none of the three is mandatory, but in practice every monthly delivery populates at least one. We walk through the first two here.

TotalReturns carries one or more `<TotalReturn>` entries (type `TotalReturnType`), each describing a total-return value over a specific calculation period, including the effect of reinvested distributions. All four of its core children are mandatory: `<Currency>`, `<Value>` (a nested element containing either an `<AbsoluteValue>` in the currency or a `<PercentageValue>` — the common case is a percentage), `<ValidityDate>` (the date at which the return is measured), and `<CalculationPeriod>` (an enumeration that names the period length). An optional `<AnnualizedFlag>` marks figures that have been annualised — as a rule, any period of a year or more is reported in annualised form so that three-year and five-year figures are directly comparable. Optional `<CalculationMethod>`, `<Investment>`, `<StartCalculationDate>`, `<CorrectionCoefficient>`, and `<ReinvestmentCoefficient>` fields let the producer carry computation provenance alongside the value itself.

The `<CalculationPeriod>` enumeration names the periods for which total returns are customarily reported:

- **1 Day** — one business day (yesterday's return)
- **1 Week** — one week
- **1 Month**, **3 Months**, **6 Months**, **YTD** — one month, three months, six months, year-to-date
- **1 Year**, **3 Years**, **5 Years**, **10 Years** — one year, three years, five years, ten years
- **Since launch** — from the share class's inception date to the current valuation date
- **Specific** — a custom period whose exact bounds are declared through `<StartCalculationDate>` and `<ValidityDate>`

A single entry for the R-EUR-ACC class's one-year return might therefore look like this:

```
<PerformanceFigures>
  <ActionCode>C</ActionCode>
  <TotalReturns>
    <TotalReturn>
      <Currency>EUR</Currency>
      <Value>
        <PercentageValue>9.4</PercentageValue>
      </Value>
      <ValidityDate>2026-03-31</ValidityDate>
      <CalculationPeriod>1 Year</CalculationPeriod>
    </TotalReturn>
    <TotalReturn>
      <Currency>EUR</Currency>
      <Value>
        <PercentageValue>6.8</PercentageValue>
      </Value>
      <ValidityDate>2026-03-31</ValidityDate>
      <CalculationPeriod>3 Years</CalculationPeriod>
      <AnnualizedFlag>true</AnnualizedFlag>
    </TotalReturn>
  </TotalReturns>
</PerformanceFigures>
```

```
</TotalReturns>
</PerformanceFigures>
```

Two structural points about this listing. First, the `<PercentageValue>` is in **human-readable per-cent**: 9.4 means 9.4%, not 0.094. The convention is the same as for every other percentage in the schema and differs from the fractional form that some numerical libraries default to. Second, the three-year entry carries `<AnnualizedFlag>true</AnnualizedFlag>`, making explicit that the 6.8% figure is a compound annual growth rate rather than a cumulative three-year return. A consumer that ignores the flag and treats the figure as a cumulative will understate the class's three-year performance by a factor of roughly three.

The **Performances** container is a near-twin of `TotalReturns` whose children (type `PerformanceType`) carry the same kind of information with slightly different shape. Its three mandatory children are `<ValidityDate>`, `<CalculationPeriod>`, and `<PercentageValue>`; optional children include `<AnnualizedFlag>`, `<CalculationMethod>`, `<StartCalculationDate>`, `<Investment>`, `<DateInterval>`, and a `<DataSeries>` block for producers that want to ship the underlying point series alongside the summary figure. In practice producers pick one of the two containers as their primary performance vehicle and use the other rarely; most pipelines we have seen populate `TotalReturns`.

The **net-versus-gross distinction** — is this return reported after fees or before them? — is *not* carried through a dedicated enumerated field in the schema. There is no `CalculationBasis` enum. Producers who need to transport the distinction do so through the free-text `<CalculationMethod>` child, using a short convention such as `Net of fees`, `includes reinvested distributions`. Consumers should read the `<CalculationMethod>` on every entry whose interpretation matters to them, and projects whose producers and consumers talk to each other should agree a project-level convention for the calculation-method text. Chapter 10 treats the validation rules that a performance-reporting pipeline should enforce, and Chapter 13 returns to the project-level convention question.

The three share classes of the Europa Growth Fund each carry their own `PerformanceFigures` block. At the close of 31 March 2026, a representative subset of the values (net of all fees, annualised where applicable) looks like this:

- **R-EUR-ACC**: 1 Year +9.4%, 3 Years +6.8% annualised, 5 Years +7.1% annualised, Since launch +6.3% annualised
- **R-CHF-ACC-HEDGED**: 1 Year +8.1%, 3 Years +5.2% annualised, 5 Years +6.0% annualised, Since launch +5.1% annualised
- **I-EUR-DIST**: 1 Year +10.2%, 3 Years +7.5% annualised, 5 Years +7.8% annualised, Since launch +6.9% annualised

Two patterns are visible in the numbers. First, the R-CHF-ACC-HEDGED class underperforms its base class by roughly one to one and a half percentage points per year, which is the typical order of magnitude for EUR/CHF hedging cost over a multi-year period. Second, the institutional class outperforms the retail class by roughly three-quarters of a percentage point per year, which is approximately the difference in management fees (0.75% for institutional versus 1.50% for retail). Neither pattern is accidental; both are expected consequences of the fee and hedging structure we described in Part II.

5.13.2 Ratings, Watermarks, Listed Prices

Beyond total returns, a handful of smaller dynamic figures deserve mention.

Ratings live in `PerformanceFigures/Ratings`, where each `<Rating>` carries a `<Note>` (the rating value as given by the agency — "5 stars", "AA+", "Silver"), optionally a `<NumericalValue>`, a `<ValidityStartDate>` and optional `<ValidityEndDate>`, and a choice between `<ListedAgency>` (drawn from an enumeration: FERI, FITCH, MOODYS, MORNINGSTAR, STANDARD-POORS) and `<UnlistedAgency>` (a `CompanyType` for other providers). A Morningstar five-star rating for the Europa Growth Fund's R-EUR-ACC class would be carried as `<Note>5</Note>` with `<ListedAgency>MORNINGSTAR</ListedAgency>`.

Dividend events for distributing share classes live in `ShareClass/Distributions`, as we mentioned in §5.10.1. Between ex-dates the class's portfolio continues to generate income; consumers that render client statements on arbitrary dates between distributions compute the accrual themselves from the portfolio income rather than reading a dedicated field, because the schema does not define one. The dividend *events* are the time-series in `Distributions`; the current accrual is derived from the portfolio (Chapter 6) and the most recent event.

Bid and ask prices apply to exchange-traded share classes — UCITS that are listed on a public market in addition to, or instead of, being traded through the fund's primary subscription/redemption process. They live inside `ShareClass/Prices/Price/OtherPrices`, alongside the mid-NAV, and are populated only for listed classes. The `ShareClass/MarketPlaces` block (§5.9.1) identifies the exchanges on which the class trades; the bid/ask, when needed, sits alongside the NAV in the price block.

High watermarks for share classes that charge a performance fee live in `ShareClass/HighWatermark`, which we already introduced in §5.10.4. The watermark is the reference NAV above which new performance fees accrue, and is typically reset annually on the fee crystallisation date or, in some structures, only when a drawdown is fully recovered. Consumers that show investors the performance-fee regime display the watermark alongside the current NAV so that the investor can see how close the class is to the next fee accrual.

Every one of these fields is optional. Producers that do not use performance fees omit the high watermark; producers whose classes are not listed omit the bid/ask; producers whose classes carry no external ratings omit the ratings block. The schema is permissive about which fields are populated, and consumers are expected to cope with partial blocks.

5.14 Umbrella Funds and Sub-Funds

We close Part IV of the chapter with a variation that we have, until now, deliberately avoided: the umbrella fund structure under which most real European UCITS actually live. Everything we have said about the Europa Growth Fund so far has treated it as a standalone fund — a single legal entity with a single portfolio, a single investment policy, and a set of share classes. In reality, a Luxembourg UCITS of its size would almost always be organised as a *sub-fund* within an *umbrella*: a single legal entity called "*Europa Asset Management Investments SICAV*" (the umbrella), inside which the Europa Growth Fund is one of several sub-funds, each with its own portfolio and each

with its own share classes. The umbrella structure is legally efficient — one set of constitutional documents, one board of directors, one regulator — while giving each sub-fund its own investment mandate and its own economic identity.

FundsXML models this structure through the `<Subfunds>` alternative inside `<Fund>` that we deferred in §5.2. An umbrella is represented as a single `<Fund>` element whose `<SingleFundFlag>` is `false` and whose `<Subfunds>` container (instead of `<SingleFund>`) holds one `<Subfund>` element per sub-fund. Each `<Subfund>` is itself a fully-fledged unit — schema type `SubfundType` — with its own `Identifiers`, `Names`, `Currency`, `SubfundStaticData`, `SubfundDynamicData`, and its own `ShareClasses` list. The umbrella's fund-level `<Identifiers>/<LEI>` identifies the umbrella SICAV as a legal entity; each `<Subfund>/<Identifiers>/<LEI>` identifies the individual sub-fund. A consumer that walks the tree can therefore distinguish umbrella and sub-fund at the LEI level without ambiguity.

Two structural consequences deserve to be named. First, the **LEI hierarchy doubles**: the umbrella has its own LEI on the outer `<Fund>`, and each sub-fund has its own LEI on its `<Subfund>`. Regulators that report at the umbrella level read the umbrella LEI; regulators that report at the sub-fund level read the sub-fund LEI. The asset manager, meanwhile, has its own separate LEI from Chapter 4, and the three should never be confused. Second, the **share classes operationally attach to the sub-fund**: the `<ShareClass>` elements live inside `<Subfund>/<ShareClasses>` rather than inside `<SingleFund>/<ShareClasses>`, but they use the same `ShareClassType` and the same vocabulary we built up in Part II. For a consumer that already understands the standalone-fund case, the step up to an umbrella is therefore small: read `<Subfunds>/<Subfund>` instead of `<SingleFund>` as the container for share classes, and everything below that level is unchanged. This chapter does not build out a complete umbrella example; Chapter 13 returns to the umbrella form in production context when it discusses the operational implications of running a pipeline over a multi-sub-fund SICAV.

For the running example of this chapter we continue to show the Europa Growth Fund as a standalone fund — a `<Fund>` with `<SingleFundFlag>true</SingleFundFlag>` and a `<SingleFund>` child rather than a `<Subfunds>` child. This is a pedagogical simplification, not a representation of its real legal form. The simplification lets us focus on the share-class structure without also having to explain umbrella mechanics; the complete example in §5.15 is in the standalone form.

5.15 The Complete Fund Element for the Europa Growth Fund

We are now ready to assemble the first complete `<Fund>` element of the book. The listing below combines everything we have treated in Parts I, II, and III of this chapter: the mandatory identity header, the full `FundStaticData` block with classifications and benchmarks, the `FundDynamicData` fund-level aggregates, the `SingleFund` container, and three `<ShareClass>` entries. To keep the listing readable we show the R-EUR-ACC class in full and collapse the other two classes to their distinguishing fields; portfolio and flow sub-blocks are elided with comments because they belong to Chapters 6 and 7. The complete listing in its fully-populated form appears in Appendix D, and the fragment below has been validated against `FundsXML4.xsd` end-to-end when embedded in the standard `<FundsXML4>/<Funds>` envelope.

```
<Fund>
  <Identifiers>
```



```

    <LEI>549300ABCDEFHIJKL34</LEI>
    <OtherID ListedType="INTERNAL FUND CODE">EAM-EGF-001</OtherID>
</Identifiers>
<Names>
    <OfficialName>Europa Asset Management Investments – Europa Growth Sub-
Fund</OfficialName>
    <MarketingName>Europa Growth Fund</MarketingName>
    <ShortName>EGF</ShortName>
    <LanguageNames>
        <Name language="de">Europa Wachstumsfonds</Name>
        <Name language="fr">Europa Fonds de Croissance</Name>
    </LanguageNames>
</Names>
<Currency>EUR</Currency>
<SingleFundFlag>true</SingleFundFlag>

<FundStaticData>
    <DomicileCountry>LU</DomicileCountry>
    <ListedLegalStructure>UCITS - SICAV</ListedLegalStructure>
    <InceptionDate>2012-01-15</InceptionDate>
    <FundTexts>
        <FundText>
            <Language>en</Language>
            <Date>2025-12-31</Date>
            <ListedType>INVESTMENT STRATEGY</ListedType>
            <Title>Investment Objective</Title>
            <Content>The Europa Growth Fund seeks long-term capital appreciation through
investment in a diversified portfolio of European equities selected for their growth
potential.</Content>
        </FundText>
        <!-- additional FundText entries in de, fr, it ... -->
    </FundTexts>
    <Classifications>
        <Classification>
            <ListedGroup>EFAMA</ListedGroup>
            <Type>EFC</Type>
            <Language>en</Language>
            <Value level="1">Equity</Value>
            <Value level="2">Equity Europe</Value>
        </Classification>
        <Classification>
            <ListedGroup>MORNINGSTAR</ListedGroup>
            <Type>Global Category</Type>
            <Language>en</Language>
            <Value>Europe Large-Cap Growth Equity</Value>
        </Classification>
    </Classifications>
    <Benchmarks>
        <Benchmark>
            <BenchmarkID>MSCI-EUR-NR</BenchmarkID>
            <Name>MSCI Europe Net Total Return EUR</Name>
            <Currency>EUR</Currency>

```

```

    <Provider>
      <Identifiers><LEI>549300YZUBM5UFHQKY25</LEI></Identifiers>
      <Name>MSCI Limited</Name>
    </Provider>
    <BenchmarkType>Market Index</BenchmarkType>
  </Benchmark>
</Benchmarks>
<FundHedgingStrategy>
  <HedgingStrategy>Full NAV hedge</HedgingStrategy>
</FundHedgingStrategy>
<OngoingCosts>
  <OngoingCost>
    <CostType>Ongoing Charges</CostType>
    <PublicationDate>2025-06-30</PublicationDate>
    <ValidFrom>2025-07-01</ValidFrom>
    <ValidTo>2026-06-30</ValidTo>
    <Percentage>1.85</Percentage>
  </OngoingCost>
</OngoingCosts>
<SFDRProductType>6</SFDRProductType>
</FundStaticData>

<FundDynamicData>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
      <TotalAssetNature>OFFICIAL</TotalAssetNature>
      <TotalNetAssetValue>
        <Amount ccy="EUR" isFundCcy="true">464552848.78</Amount>
      </TotalNetAssetValue>
    </TotalAssetValue>
  </TotalAssetValues>
  <!-- Portfolios: see Chapter 6. Benchmarks: dynamic level values, elided. -->
</FundDynamicData>

<SingleFund>
  <SingleFundStaticData>
    <ExchangeTradedFlag>false</ExchangeTradedFlag>
    <FundOfFundFlag>false</FundOfFundFlag>
    <SocialResponsibleFlag>false</SocialResponsibleFlag>
    <ManagementType>ACTIVE</ManagementType>
    <PricePublishedFlag>true</PricePublishedFlag>
    <SalesCategory>PUBLIC</SalesCategory>
  </SingleFundStaticData>
  <ShareClasses>
    <ShareClass>
      <Identifiers>
        <ISIN>LU2100000011</ISIN>
        <GermanWKN>A2GROW</GermanWKN>
        <SwissValorenCode>54321001</SwissValorenCode>
      </Identifiers>
      <Names>

```

```

    <OfficialName>Europa Growth Fund – R EUR ACC</OfficialName>
    <ShortName>EGF R EUR ACC</ShortName>
  </Names>
  <Currency>EUR</Currency>
  <ShareClassType>
    <Code>R</Code>
    <EarningUse>R</EarningUse>
    <Name>Retail Accumulating</Name>
  </ShareClassType>
  <InceptionDate>2012-01-15</InceptionDate>
  <RegistrationCountries>
    <RegistrationCountry><CountryCode>LU</CountryCode><Status>Registered</Status>
  </RegistrationCountry>
    <RegistrationCountry><CountryCode>DE</CountryCode><Status>Registered</Status>
  </RegistrationCountry>
    <RegistrationCountry><CountryCode>FR</CountryCode><Status>Registered</Status>
  </RegistrationCountry>
    <!-- IT, ES, NL, AT, BE, PT, CH, SE elided -->
  </RegistrationCountries>
  <SubscriptionRestrictions>
    <MinSubscriptionAmount>
      <Amount ccy="EUR">100</Amount>
    </MinSubscriptionAmount>
    <MinSubscriptionAmountSubsequent>
      <Amount ccy="EUR">50</Amount>
    </MinSubscriptionAmountSubsequent>
  </SubscriptionRestrictions>
  <OpenToNewInvestorsFlag>true</OpenToNewInvestorsFlag>
  <CurrencyHedgedFlag>false</CurrencyHedgedFlag>
  <IsReferenceShareClass>true</IsReferenceShareClass>
  <IsPublicFlag>true</IsPublicFlag>
  <Prices>
    <Price>
      <ActionCode>C</ActionCode>
      <NavDate>2026-03-31</NavDate>
      <PriceCurrency>EUR</PriceCurrency>
      <PriceNature>OFFICIAL</PriceNature>
      <NavPrice>124.50780000</NavPrice>
    </Price>
  </Prices>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
      <TotalAssetNature>OFFICIAL</TotalAssetNature>
      <TotalNetAssetValue>
        <Amount ccy="EUR" isShareClassCcy="true">153712654.83</Amount>
      </TotalNetAssetValue>
      <SharesOutstanding>1234567</SharesOutstanding>
      <Ratio>33.09</Ratio>
    </TotalAssetValue>
  </TotalAssetValues>
  <Fees>

```

```

    <Fee>
      <Type>Management Fee</Type>
      <PayReceive>P</PayReceive>
      <Maximum>1.50</Maximum>
      <CalculationMethod>Per annum of average net assets, accrued
daily</CalculationMethod>
      <DataByPeriods>
        <DataByPeriod>
          <BeginDate>2025-07-01</BeginDate>
          <EndDate>2026-06-30</EndDate>
          <CurrentDate>2026-03-31</CurrentDate>
          <Values><FeeAsPercentageOfTNA>1.50</FeeAsPercentageOfTNA></Values>
        </DataByPeriod>
      </DataByPeriods>
    </Fee>
  </Fees>
  <PerformanceFigures>
    <ActionCode>C</ActionCode>
    <TotalReturns>
      <TotalReturn>
        <Currency>EUR</Currency>
        <Value><PercentageValue>9.4</PercentageValue></Value>
        <ValidityDate>2026-03-31</ValidityDate>
        <CalculationPeriod>1 Year</CalculationPeriod>
      </TotalReturn>
      <TotalReturn>
        <Currency>EUR</Currency>
        <Value><PercentageValue>6.8</PercentageValue></Value>
        <ValidityDate>2026-03-31</ValidityDate>
        <CalculationPeriod>3 Years</CalculationPeriod>
        <AnnualizedFlag>true</AnnualizedFlag>
      </TotalReturn>
    </TotalReturns>
  </PerformanceFigures>
</ShareClass>

<ShareClass>
  <Identifiers>
    <ISIN>LU2100000029</ISIN>
    <GermanWKN>A2GRCH</GermanWKN>
    <SwissValorenCode>54321002</SwissValorenCode>
  </Identifiers>
  <Names>
    <OfficialName>Europa Growth Fund – R CHF ACC Hedged</OfficialName>
    <ShortName>EGF R CHF ACC H</ShortName>
  </Names>
  <Currency>CHF</Currency>
  <ShareClassType>
    <Code>R</Code><EarningUse>R</EarningUse>
    <Name>Retail Accumulating, Hedged</Name>
  </ShareClassType>
  <InceptionDate>2018-06-01</InceptionDate>

```

```

<SubscriptionRestrictions>
  <MinSubscriptionAmount><Amount ccy="CHF">100</Amount></MinSubscriptionAmount>
</SubscriptionRestrictions>
<CurrencyHedgedFlag>true</CurrencyHedgedFlag>
<FundHedgingStrategy>
  <HedgingStrategy>Full NAV hedge</HedgingStrategy>
</FundHedgingStrategy>
<IsPublicFlag>true</IsPublicFlag>
<Prices>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-31</NavDate>
    <PriceCurrency>CHF</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>119.20110000</NavPrice>
  </Price>
</Prices>
<TotalAssetValues>
  <TotalAssetValue>
    <NavDate>2026-03-31</NavDate>
    <TotalAssetNature>OFFICIAL</TotalAssetNature>
    <TotalNetAssetValue>
      <Amount ccy="CHF" isShareClassCcy="true">54438456.22</Amount>
    </TotalNetAssetValue>
    <SharesOutstanding>456789</SharesOutstanding>
    <Ratio>12.21</Ratio>
  </TotalAssetValue>
</TotalAssetValues>
<!-- Fees identical in structure to R-EUR-ACC, elided.
      PerformanceFigures: own series, elided. -->
</ShareClass>

<ShareClass>
  <Identifiers>
    <ISIN>LU2100000037</ISIN>
    <GermanWKN>A2GRIN</GermanWKN>
  </Identifiers>
  <Names>
    <OfficialName>Europa Growth Fund – I EUR DIST</OfficialName>
    <ShortName>EGF I EUR DIST</ShortName>
  </Names>
  <Currency>EUR</Currency>
  <ShareClassType>
    <Code>I</Code><EarningUse>D</EarningUse>
    <Name>Institutional Distributing</Name>
  </ShareClassType>
  <InceptionDate>2020-09-15</InceptionDate>
  <SubscriptionRestrictions>
    <MinSubscriptionAmount><Amount ccy="EUR">1000000</Amount>
  </MinSubscriptionAmount>
  </SubscriptionRestrictions>
  <CurrencyHedgedFlag>>false</CurrencyHedgedFlag>

```

```

<IsPublicFlag>true</IsPublicFlag>
<Prices>
  <Price>
    <ActionCode>C</ActionCode>
    <NavDate>2026-03-31</NavDate>
    <PriceCurrency>EUR</PriceCurrency>
    <PriceNature>OFFICIAL</PriceNature>
    <NavPrice>108.33440000</NavPrice>
  </Price>
</Prices>
<TotalAssetValues>
  <TotalAssetValue>
    <NavDate>2026-03-31</NavDate>
    <TotalAssetNature>OFFICIAL</TotalAssetNature>
    <TotalNetAssetValue>
      <Amount ccy="EUR" isShareClassCcy="true">254115322.56</Amount>
    </TotalNetAssetValue>
    <SharesOutstanding>2345678</SharesOutstanding>
    <Ratio>54.70</Ratio>
  </TotalAssetValue>
</TotalAssetValues>
<Distributions>
  <Distribution>
    <ActionCode>C</ActionCode>
    <DividendStatus>OFFICIAL</DividendStatus>
    <AnnouncementDate>2025-05-02</AnnouncementDate>
    <RecordDate>2025-05-14</RecordDate>
    <ExDate>2025-05-15</ExDate>
    <PaymentDate>2025-05-22</PaymentDate>
    <PaymentCurrency>EUR</PaymentCurrency>
    <GrossDividendAmount>
      <PerShare><Amount ccy="EUR">2.1450</Amount></PerShare>
    </GrossDividendAmount>
    <NetDividendAmount>
      <PerShare><Amount ccy="EUR">1.8450</Amount></PerShare>
    </NetDividendAmount>
  </Distribution>
</Distributions>
  <!-- Fees: Management Fee 0.75%, same structure as R-EUR-ACC, elided. -->
</ShareClass>
</ShareClasses>
</SingleFund>
</Fund>

```

Reading the block is best done in three passes, matching the three parts of this chapter.

The **first pass** looks at the fund's mandatory header and its static data. The `<Identifiers>` block carries the fund's LEI (549300ABCDEFGHIJKL34), identifying the Europa Growth Fund as a legal entity distinct from the LEI of its asset manager (Chapter 4), together with a producer-internal reference code in an `OtherID` entry. The `<Names>` block carries the fund's full official name, its marketing name, a short code, and translations into German and French. The fund's base currency is EUR and the `<SingleFundFlag>` is `true`. Inside `<FundStaticData>`

the domicile is Luxembourg; the legal structure is *UCITS - SICAV*; the fund was launched in 2012; investment-strategy text is carried under `<FundTexts>`; two parallel `<Classifications>` entries — EFAMA and Morningstar — describe the fund as a European equity product; a single `<Benchmark>` names the MSCI Europe Net Total Return EUR index; a single `<OngoingCost>` entry carries the fund-level headline ongoing charges figure of 1.85% for the current fiscal year; and the `<SFDRProductType>` is 6, marking the fund as an Article 6 product.

The **second pass** looks at the share-class structure, carried inside `<SingleFund>`. The `<SingleFundStaticData>` block labels the fund as actively managed, not exchange-traded, not a fund of funds, and classified as PUBLIC sales. The three share classes themselves are R-EUR-ACC, R-CHF-ACC-HEDGED, and I-EUR-DIST, each inside its own `<ShareClass>` element. Each carries its own `<Identifiers>` block with ISIN and adjacent codes, its own `<Names>` block, its own `<Currency>` (EUR or CHF), its own `<ShareClassType>` with `<Code>` and `<EarningUse>` (R for reinvesting, D for distributing), its own `<InceptionDate>`, and its own subscription minimum. The R-CHF-ACC-HEDGED class is the only one with `<CurrencyHedgedFlag>true</CurrencyHedgedFlag>`, and it carries a `<FundHedgingStrategy>Full NAV hedge</FundHedgingStrategy>` at class level in addition. The I-EUR-DIST class is the only one with `<EarningUse>D</EarningUse>` and is therefore the only one with a populated `<Distributions>` block — showing the 15 May 2025 dividend event in both its gross and net form.

The **third pass** looks at the dynamic figures. Each `<ShareClass>` carries its own `<Prices>/<Price>` entry — the NAV per unit at 31 March 2026 in the class's own currency — and its own `<TotalAssetValues>/<TotalAssetValue>` with the mandatory `<NavDate>` and `<TotalAssetNature>`, the class's total net asset value (EUR or CHF, marked with `isShareClassCcy="true"`), the shares outstanding, and a `<Ratio>` giving the class's share of the fund's total net assets. Alongside the per-class figures, `<FundDynamicData>/<TotalAssetValues>` carries a single fund-level aggregate entry: 464,552,848.78 EUR, marked with `isFundCcy="true"` to make clear that the currency is the fund's base currency after conversion of the CHF-denominated class. The `<PerformanceFigures>` block on R-EUR-ACC shows two total-return entries (one-year and three-year) to illustrate the shape; a fully populated class would carry six to eight entries covering the full set of standard periods.

This `<Fund>` element is the centrepiece of the Europa Growth Fund's month-end delivery for 31 March 2026, and it will appear — in its complete form, with the portfolio and flows fully populated — in Appendix D. Every subsequent chapter of Part II will take one fragment of the same element for detailed treatment, so that by the time we reach the end of Chapter 9 the reader has seen every part of it explained at least once.

5.16 Common Pitfalls

The following short list captures the mistakes that, in our experience, cause the greatest share of fund-element-related production incidents.

- **The ISIN is placed on the fund's `<Identifiers>` instead of on each share class's `<Identifiers>`.** Distributors consuming the delivery fail to distinguish the classes and route subscription orders to whichever class they find first. The fix is to move the ISIN into the `<Identifiers>` block of each `<ShareClass>` where it belongs.

- **The share-class currency is confused with the fund base currency.** A consumer reads 119.2011 from the R-CHF-ACC-HEDGED class and treats it as a EUR number, producing NAV values that are wrong by the EUR/CHF rate. The fix is to respect the `ccy` attribute on every `<Amount>` element, and to use the optional `isFundCcy` / `isShareClassCcy` attributes as additional self-description.
- **A share class is omitted from a delivery.** The consumer's current-state engine interprets the absence as a discontinuation and marks the class as closed, leading to factsheet errors and failed order routing. The rule is that every share class must appear in every full delivery even if its data has not changed since the last one.
- **The `EarningUse` code is the wrong letter.** The schema uses `R` for *reinvesting* (accumulating) and `D` for *distributing*; producers occasionally get them the wrong way round because *R* intuitively looks like *retail* or *return*. A class labelled `<EarningUse>R</EarningUse>` when it should be `D` causes every distributing-class consumer to misclassify the fund. The fix is to verify the letter on every class against the class's prospectus behaviour.
- **Performance figures are emitted without a calculation-method annotation.** The schema does not carry a net/gross flag as a structured field; the distinction rides in the free-text `<CalculationMethod>` child of `<TotalReturn>`. One downstream consumer treats the figures as net, another as gross, and the two disagree about how the fund is performing. The fix is to populate a short, project-stable calculation-method string on every entry and to document the convention at project level.
- **The inception date is inherited from the fund rather than set per share class.** A *since-launch* performance figure for the R-CHF-ACC-HEDGED class starts in 2012 (fund inception) rather than 2018 (class inception), showing six years of performance that the class did not experience. The fix is to populate `<InceptionDate>` inside each `<ShareClass>` and compute performance from there.
- **A consumer parses the share-class name to infer currency and earning use.** The consumer's logic breaks when it meets an asset manager whose naming convention differs. The fix is to read `ShareClass/Currency`, `ShareClass/ShareClassType/EarningUse`, `ShareClass/CurrencyHedgedFlag`, and `ShareClass/SubscriptionRestrictions/MinSubscriptionAmount` as structured fields and never depend on the name.
- **The fund-level `<Classifications>` block is used to transport ad-hoc strings rather than recognised classification systems.** A producer writes `<ListedGroup>EFAMA</ListedGroup>` but populates `<Value>` with a home-grown label that the EFAMA taxonomy does not use. Consumers that recognise EFAMA then mismatch. The fix is to populate each `<Classification>` with values from the taxonomy its `<ListedGroup>` names, and to use `<UnlistedGroup>` when the classification is proprietary.

5.17 Key Takeaways

- A FundsXML document may contain one or several `<Fund>` elements. Each `<Fund>` begins with four mandatory children — `Identifiers`, `Names`, `Currency`, `SingleFundFlag` — and then optionally carries `FundStaticData`, `FundDynamicData`, and a choice of `SingleFund` (for non-umbrella funds) or `Subfunds` (for umbrella structures). Share classes live inside `SingleFund/ShareClasses`, not inside `FundStaticData`.

- Fund identifiers live at two levels, both using the same `IdentifiersType` container. `Fund/Identifiers` carries fund-level codes (principally `LEI`); each `ShareClass/Identifiers` carries share-class-level codes (`ISIN`, `GermanWKN`, `SwissValorenCode`, `CUSIP`, `SEDOL`, and proprietary `OtherID` entries). Confusing the levels is the commonest structural mistake in the fund element.
- Names are modelled through the `NamesType` container with a mandatory `<OfficialName>`, optional `<MarketingName>`, `<ShortName>`, `<FullName>`, `<PreviousName>`, and a `<LanguageNames>` child that holds the multilingual translations as `<Name language="xx">` entries. Longer narrative text lives in `FundStaticData/FundTexts`, with a `<ListedType>` that names the kind of text and a free-form `<Content>`.
- Fund classification — domicile, legal structure, inception date, investment classification, benchmark, SFDR product type — lives in `FundStaticData`. Investment classification is expressed through `Classifications/Classification` entries that reference external taxonomies (EFAMA, Morningstar, MiFID, and others) rather than a single closed vocabulary. The regulatory templates in Chapter 8 provide richer versions of the same information for regulated use cases.
- Share classes are the structural branch of the fund element. Fields that differ between classes — identifiers, `Currency`, `ShareClassType/EarningUse`, `CurrencyHedgedFlag`, `SubscriptionRestrictions`, per-class `Fees`, `InceptionDate`, `Prices`, `TotalAssetValues`, `PerformanceFigures`, `RegistrationCountries` — live on the individual `<ShareClass>` element. Everything common to all classes stays on `<Fund>` or in `FundStaticData`.
- Fees are modelled at three layers. `FundStaticData/OngoingCosts` carries headline fund-level aggregate figures (TER, ongoing charges, transaction costs, performance fees) with `<Percentage>` in human-readable per-cent. `ShareClass/Fees/Fee` carries per-class management and distribution fees with mandatory `Type` and `PayReceive`, optional `Minimum/Maximum/CalculationMethod`, and `DataByPeriods/DataByPeriod/Values/FeeAsPercentageOfTNA` for the period values. `ShareClass/HighWatermark` carries the high-watermark reference for performance-fee classes.
- Per-class dynamic data lives inside each `<ShareClass>`: `Prices` for the NAV per unit, `TotalAssetValues` for the class aggregate with shares outstanding and ratio, `Distributions` for dividend events, `Flows` for subscriptions and redemptions, `PerformanceFigures` for total returns and ratings. `FundDynamicData/TotalAssetValues` carries only the single fund-level aggregate; consumers that need per-class dynamic figures read them from the classes themselves.
- Performance is carried in `ShareClass/PerformanceFigures` with `<TotalReturn>` entries that name their calculation period through a `<CalculationPeriod>` enumeration (`1 Day`, `1 Month`, `YTD`, `1 Year`, `3 Years`, `Since launch`, `Specific`) and carry their value as either `<AbsoluteValue>` or `<PercentageValue>`. The net-versus-gross distinction rides in free-text `<CalculationMethod>`; there is no dedicated enum for it in the schema.
- Umbrella funds are modelled through the `<Subfunds>/<Subfund>` alternative inside `<Fund>`, with each sub-fund carrying its own `Identifiers`, `Names`, `Currency`, `SubfundStaticData`, `SubfundDynamicData`, and `ShareClasses`. The running example in §5.15 is shown in the standalone `<SingleFund>` form for pedagogical clarity; Chapter 13 treats the umbrella form in production context.

We have now described the Europa Growth Fund: what it is, how its share classes differ, and what its NAVs look like on 31 March 2026. What we have not yet seen is what the fund actually holds. Chapter 6 opens the `Portfolios` substructure of `FundDynamicData` and walks, asset class by asset class, through the instruments that make up the roughly two hundred positions on the fund's books.



Portfolio and Positions

Holdings and asset classes

6.1 Setting the Scene: What the Fund Actually Holds

At the end of Chapter 5 we had a complete description of the Europa Growth Fund: its identity, its three share classes, its fees, its total net assets of 464,552,848.78 EUR on 31 March 2026. What we did not have was any knowledge of *how that money is invested*. The `Portfolios` substructure of `FundDynamicData`, which Chapter 5 left as a placeholder comment, is where the answer lives; and alongside it, outside the `<Funds>` container at the root level of the document, sits its indispensable companion, `AssetMasterData`. This chapter treats both.

It is the longest chapter in the book — forty-five pages — and it is long for three reasons. First, the portfolio is structurally the richest part of a FundsXML delivery: it carries every instrument the fund holds, one entry per position, and instruments come in seven or eight broad families (equities, bonds, funds, derivatives, cash, real estate, commodities, alternatives) each with their own specialised fields. Second, the linkage between `Portfolio` and `AssetMasterData` through `UniqueID` is the working expression of design principle 2 from Chapter 3 — *linkage by UniqueID, not by nesting* — and mastering that linkage is central to every subsequent chapter. Third, portfolio breakdowns, the pre-computed aggregates that feed fact sheets and regulatory reports, have their own structure and their own conventions. All three topics justify the space.

One editorial decision deserves to be stated up front. The Europa Growth Fund, as we have seen, is an actively managed European equity UCITS. It holds roughly 80 to 150 equity positions, a handful of cash accounts in several currencies, and the two currency forwards that implement the CHF hedge for the R-CHF-ACC-HEDGED share class. That is what its real portfolio looks like, and it is what the complete end-to-end example in §6.12 will show. For the asset classes the fund does not hold — bonds, investment funds held as positions, real estate, commodities — the per-asset-class sections use their own isolated examples drawn from generic illustrative funds, and the complete example in §6.12 does not attempt to force these classes into the Europa Growth Fund artificially. The reader who wants to see a bond position renders §6.7's example; the reader who wants to see a derivative in context renders §6.9.4 and the corresponding fragment of §6.12. The integrity of the running example and the coverage of the asset classes can both be preserved without compromise.

By the end of this chapter, you should be able to:

- explain why portfolio positions reference `AssetMasterData` entries rather than containing them;
- produce a valid `UniqueID` that links a position to its asset description and defend your choice of generator;
- recognise the common fields that appear on every position, regardless of asset class;
- populate the asset-class-specific fields for equities, bonds, funds, derivatives, cash, and real estate;
- distinguish a derivative's notional from its market value and never again confuse the two;

- compute portfolio breakdowns by sector, country, currency, rating, and maturity from a list of positions, and recognise where breakdowns legitimately differ from a naive recomputation;
- read a complete mixed portfolio of a European equity UCITS on sight and cross-reference every position to its `AssetMasterData` entry.

The chapter is organised into five parts. **Part I** (§6.2 and §6.3) establishes the shape of the problem and the linkage mechanism. **Part II** (§6.4 and §6.5) treats the common fields that appear on every position and every asset. **Part III** (§6.6 through §6.10) walks through the asset classes one at a time, each with its own specialised fields and a small isolated example. **Part IV** (§6.11) covers portfolio breakdowns. **Part V** (§6.12 through §6.14) assembles the complete portfolio of the Europa Growth Fund, lists the common pitfalls, and points at Chapter 7.

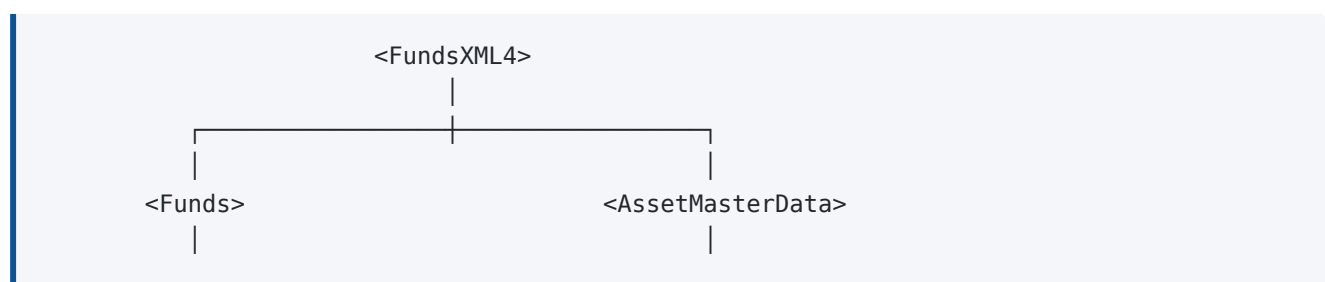
6.2 Portfolios and AssetMasterData — The Shape of the Problem

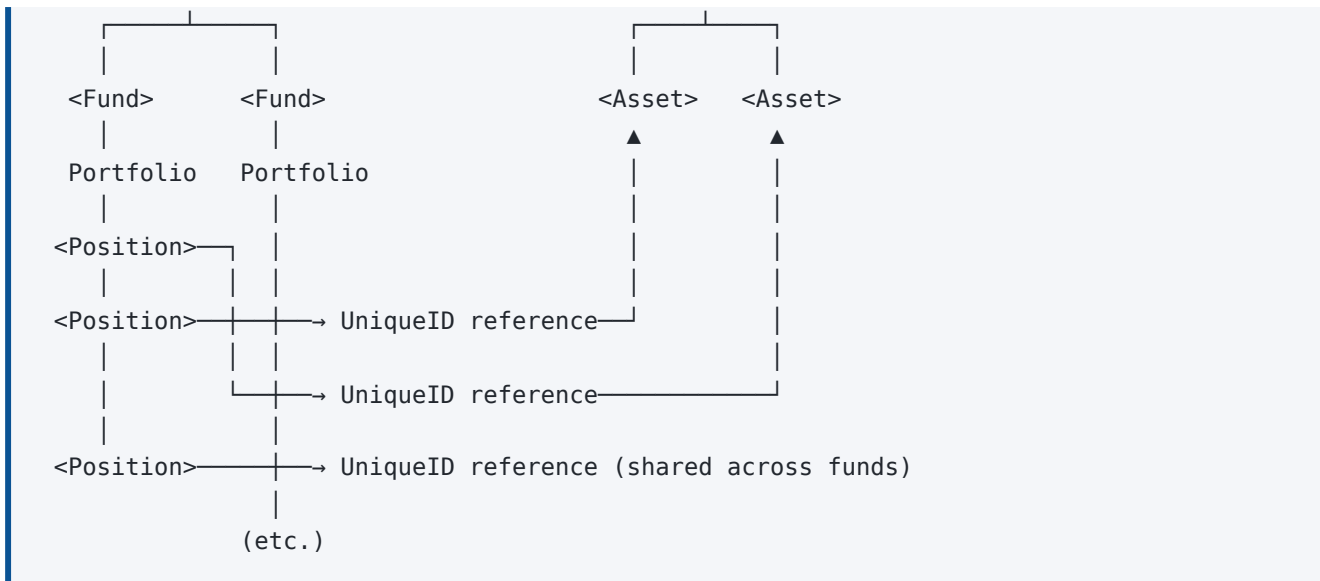
Before we look at any field, a step back. The portfolio representation in FundsXML is split into two physically separate blocks — `<Portfolios>` inside each fund's `<FundDynamicData>`, and `<AssetMasterData>` at the root of the document — and the split is the first thing a reader of a FundsXML document notices and the first thing that surprises them. Why not put the full asset description inside each position, as simpler formats do? The answer is the working expression of design principle 2 from Chapter 3, and the motivation is worth making concrete.

Imagine a fund administrator's nightly batch file: twenty funds, each with a hundred and fifty positions, all serialised into a single FundsXML document. Many of those positions reference the same instruments. Siemens is held by twelve of the twenty funds; SAP by eight; Nestlé by fifteen; the short list of European blue chips appears in almost every one of them. If the full asset description of Siemens — its name, its LEI, its GICS classification, its primary listing information, its forty or so static fields — were embedded in every position that referenced it, Siemens would be described twelve times in a single file, each time with the same forty fields, each time vulnerable to a transcription or classification mismatch between copies. A producer that rewrites one copy and forgets another creates a file in which two Siemens entries disagree about what Siemens *is*, and every downstream consumer that tries to reconcile them has a data-quality problem.

The solution is the one every relational database uses: describe each instrument *once*, in a central location, and have each position refer to it through an identifier. That central location is `<AssetMasterData>`, and it sits at the root level of the FundsXML document, outside the `<Funds>` container, precisely so that it can be shared between several funds in the same delivery. Each position in each fund's portfolio carries only a short pointer — a `UniqueID` — that tells the consumer where to find the full description. The file is smaller, the descriptions are consistent by construction, and the operational discipline is easier to enforce.

Figure 6.1 — The portfolio/master-data split





The figure makes three points. First, both funds in the delivery reach into the same `<AssetMasterData>` block, and a single asset entry can be referenced by positions in several funds simultaneously. Second, the reference is *backward* — from position to asset — not forward; there is no "list of holders" on the asset side. Third, the containment hierarchy inside each fund (Portfolio → Position) and the containment hierarchy inside the root (AssetMasterData → Asset) are parallel, and the `UniqueID` is the only connection between them.

One clarification deserves to be made now, before it becomes a source of confusion later in the chapter: **AssetMasterData exists even when the delivery contains a single fund.** The Europa Growth Fund is delivered in its own FundsXML file each month-end, not as part of a twenty-fund batch, and yet every one of its deliveries carries a complete `<AssetMasterData>` block. The reason is not architectural consistency for its own sake; the reason is that the separation keeps positions slim. A Siemens position in the Europa Growth Fund's portfolio contains eight or nine fields (the `UniqueID` reference, the valuation currency, the single `TotalValue` in fund currency, the percentage weight, the price date, plus the equity-specific sub-block carrying the unit count, the price and the market value); the Siemens description in `AssetMasterData` contains forty or more fields covering identifiers, issuer, listing, classifications, ratings, and country-specific extensions. If the two were merged, a portfolio of 150 positions would carry $150 \times 40 = 6,000$ fields of description when $150 \times 9 + 150 \times 40$ (one description per distinct asset, which, with overlap across funds, is often much smaller) would suffice. Even for a single fund, the separation is operationally meaningful.

Two architectural observations close this section. `<Portfolios>` lives inside `<FundDynamicData>` — it belongs to a specific valuation date and is re-emitted whenever a new portfolio snapshot is produced. `<AssetMasterData>` lives at the root level and applies to the entire delivery. The first is *dynamic* in the sense that its content changes every valuation day; the second is closer to *static*, even though in practice it also evolves as new instruments enter the fund's universe. Finally, **the two structures must be read together.** A portfolio position without its corresponding asset entry is meaningless; an asset entry with no position pointing at it is dead weight. The schema enforces the first half of this discipline directly at parse time — the `<UniqueID>` on each `<Asset>` is declared as `xs:ID` and the `<UniqueID>` on each `<Position>` as `xs:IDREF`, which means a position whose identifier does not match any asset is rejected by any conformant XML parser before any business-level validation even begins. The second half — orphaned asset entries — is not enforced by the schema and shows up only as a data-quality issue. Chapter 10 will formalise both halves as business-validation rules; for this chapter it is enough to carry the pairing in mind at every step.

6.3 The UniqueID Linkage in Detail

The `UniqueID` is the small glue that binds the two sides together. We treat it in three short subsections.

6.3.1 What a UniqueID Is

A `UniqueID` is a string, typically twenty to forty characters, that uniquely identifies each `<Asset>` entry inside `<AssetMasterData>` within a single FundsXML delivery. The schema does not declare it as a plain string — it declares it as `xs:ID` on the asset side and `xs:IDREF` on the position side. The practical consequence is twofold. First, the parser checks at load time that every position's `UniqueID` resolves to exactly one asset entry's `UniqueID`, and a mismatch is a schema-validation error rather than a business-rule error. Second, the value must conform to the XML Name syntax: it must begin with a letter or underscore, must not contain whitespace, and must be unique across the whole document. ISINs such as `DE0007236101` satisfy the syntax because they begin with a letter; purely numeric codes such as a raw five-digit internal reference would not, and producers using numeric internal codes typically prefix them with a letter (`EGF-12345`) to make them valid XML names.

The scope of uniqueness is important: unlike the `UniqueDocumentID` of Chapter 4, which must be unique across every delivery a producer has ever emitted, the `UniqueID` of an asset need only be unique within the file in which it appears. Two deliveries from the same producer may carry different `UniqueID` values for the same Siemens entry without violating anything, as long as each file is internally consistent. In practice, producers that care about cross-delivery consistency pick a generator that is stable over time (see §6.3.2), but this is a convention, not a schema requirement.

6.3.2 How to Generate a UniqueID

Two strategies dominate production usage, and each has its own strengths.

The first strategy is **to use the instrument's ISIN as its `UniqueID`**. The ISIN is already globally unique in the instrument universe, is already present on almost every asset the fund is likely to hold, and is human-readable enough that a debugger looking at a portfolio dump recognises each line at a glance. A position that references `DE0007236101` and an asset entry with `UniqueID=DE0007236101` are visibly the same instrument; no translation is required. The strategy is so natural that it is the default in most production pipelines.

The weakness of the ISIN strategy is that not every asset has an ISIN. Cash positions do not; over-the-counter derivatives do not; direct real estate holdings do not; some exotic private-equity commitments do not. For these assets, the producer needs a fallback, and that fallback is usually a UUID: a one hundred and twenty-eight-bit random or deterministically-derived identifier, long enough that collisions across producers are astronomically unlikely. The weakness of a UUID is the loss of readability; a debugger sees `7f8e3a2d-...` instead of a meaningful name and must perform a mental lookup.

The recommended convention, which Chapter 13 will formalise as a project-level decision, is a hybrid: **ISIN when the asset has one, UUID when it does not, and the choice must be deterministic across deliveries**. Deterministic means that the same Siemens asset, generated by the same producer on a different day, receives the same `UniqueID`. For ISIN-bearing assets this is automatic; for UUID-generated assets it requires a UUID v5 (name-

based) computed from a stable internal identifier rather than a fresh random v4 on every run. A producer that emits fresh UUIDs every day makes it impossible for consumers to reconcile the same instrument across consecutive deliveries, which is not quite a correctness problem but is a significant operational nuisance.

6.3.3 The Linkage Mechanism in Action

A concrete pair of elements makes the linkage visible. The Europa Growth Fund holds 50,000 shares of Siemens AG. In the portfolio block, inside `Fund/FundDynamicData/Portfolios/Portfolio/Positions`, the position looks like this:

```
<Position>
  <UniqueID>DE0007236101</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">8125000.00</Amount>
  </TotalValue>
  <TotalPercentage>1.75</TotalPercentage>
  <PriceDate>2026-03-31</PriceDate>
  <Equity>
    <Units>50000</Units>
    <Price><Amount ccy="EUR">162.50</Amount></Price>
    <MarketValue>
      <Amount ccy="EUR" isFundCcy="true">8125000.00</Amount>
    </MarketValue>
  </Equity>
</Position>
```

Elsewhere in the same document, inside the root-level `<AssetMasterData>`, the corresponding asset entry carries the matching `UniqueID` and the full instrument description:

```
<Asset>
  <UniqueID>DE0007236101</UniqueID>
  <Identifiers>
    <ISIN>DE0007236101</ISIN>
  </Identifiers>
  <Currency>EUR</Currency>
  <Country>DE</Country>
  <Name>Siemens AG</Name>
  <AssetType>EQ</AssetType>
  <AssetDetails>
    <Equity>
      <StockMarket>XETR</StockMarket>
      <Issuer>
        <Identifiers>
          <LEI>W38RGI023J3WT1HWRP32</LEI>
        </Identifiers>
        <Name>Siemens Aktiengesellschaft</Name>
        <BusinessCountry>DE</BusinessCountry>
```

```

        <Type>Non-financial corporations</Type>
    </Issuer>
    <Listing>A</Listing>
</Equity>
</AssetDetails>
</Asset>

```

Four structural points about this pair are worth stating before we look at any single field. First, the position has a mandatory `<TotalValue>` (typed `FundAmountType`) that carries the consolidated value of the whole position in the fund's base currency. This is the one number against which the sum of all positions must reconcile to the fund's total net assets; the detail fields inside the asset-class-specific `<Equity>` child are there for transparency, not to replace the total. Second, the asset-class-specific child — here `<Equity>` — is one of a choice of twenty-one variants: `Equity`, `Bond`, `ShareClass`, `Warrant`, `Certificate`, `Option`, `Future`, `FXForward`, `Swap`, `Repo`, `FixedTimeDeposit`, `CallMoney`, `Account`, `Fee`, `RealEstate`, `REIT`, `Loan`, `Right`, `Commodity`, `PrivateEquity`, and `CommercialPaper`. Exactly one child of the choice is required at the end of every position. (The asset-master side of the linkage — `AssetDetails` — has two additional choice members, `Index` and `Crypto`, because the schema allows those instruments to be described as assets but not to be held as portfolio positions. §6.5 returns to this asymmetry.) Third, the asset side uses `<AssetType>` as a two-character discriminator code — `EQ` for Equity, `B0` for Bond, `SC` for fund share class, `FX` for FX forward, `AC` for account, and so on — and the asset's detailed fields live inside a separate `<AssetDetails>` child whose content mirrors the position-side choice by element name. Fourth, the `ISIN` lives inside `<Identifiers>`, not as a direct child of `<Asset>`; the container takes exactly the same shape as the fund-level and share-class-level `<Identifiers>` blocks of Chapter 5.

A consumer that reads the position finds `DE0007236101` in the `UniqueID` field, jumps to the `<AssetMasterData>` block, finds the matching `<Asset>` entry, and now knows everything it needs to know about Siemens as an instrument. It does this lookup once per distinct asset per delivery, caches the results in memory, and processes the whole portfolio without ever re-parsing the asset descriptions. The cost is linear in the number of distinct instruments, not in the number of position-instrument references.

The key rule: consumers never read `Portfolio` and `AssetMasterData` in isolation. Both structures together form the complete information about what the fund holds. A production pipeline that parses only one of the two is fundamentally broken, and any validation discipline (Chapter 10) treats the pair as atomic.

Two validation rules flow directly from the linkage. First, **every `UniqueID` in a `Position` must match exactly one `<Asset>` in `AssetMasterData`**. As we saw in §6.3.1, the schema enforces this at the parser level through its `xs:ID / xs:IDREF` pairing; a position whose `UniqueID` has no target is a schema-validation failure, not a business-rule failure, and the document does not load at all. Second, **every `<Asset>` in `AssetMasterData` should be referenced by at least one `Position`**. An orphaned asset entry is harmless from a consumer's perspective but is almost always a sign of a sloppy producer — typically the residue of a position that was deleted from the producer's database without a corresponding cleanup of the asset master. Chapter 10 lists this rule among the business-validation checks that a production consumer pipeline should run on every incoming file.

6.4 Position-Level Fields

We now descend into the fields. The common position-level fields, which appear on every `<Position>` element regardless of what asset class the position belongs to, are the subject of this section. Asset-class-specific extras come later in Part III.

Table 6.1 lists the common children of `<Position>` in the order they must appear.

Table 6.1 — Common children of `<Position>`

Child element	Schema type	Purpose	Example value
<Unique ID>	xs:IDREF (max 256 chars)	Reference to the matching <Asset> in <AssetMasterData>	DE0007236101
<Identifiers>	IdentifiersType	Optional copy of key instrument codes — redundant with the asset's own Identifiers , present in some pipelines for debugging	—
<Currency>	ISOCurrencyCode-Type	Valuation currency of the instrument (its native trading currency, not the fund's base currency)	EUR
<TotalValue>	FundAmountType	The consolidated value of the whole position, in the fund's base currency. Mandatory.	<Amount ccy="EUR">8125000.00</Amount>
<OtherTotalValues>	container	Additional valuations (Hold-To-Maturity, Swing-adjusted, Market, ...)	—
<TotalPercentage>	PercentageType	The position's weight in the fund, as a percentage of total net assets	1.75
<AveragePurchasePrice>	FundAmountType	The weighted-average cost basis of the position across all buys	—
<AveragePurchaseFXRate>	FXRatesType	The weighted-average FX rate at which the position was entered	—
<Exposures>	ExposureType	Exposure values under different regulatory approaches (commitment, AIFMD)	—
<FXRates>	FXRatesType	FX rates used in evaluating this position	—
<PriceDate>	xs:date	The date of the price used for this valuation	2026-03-31
<PricingSource>	choice(Listed / Unlisted)	Pricing source — either a listed enumeration (BLOOMBERG , REUTERS , FACTSET , MARKIT , BARCLAYS , JPMORGAN , MERRILL , BROKERS , LPC , NTRS) or free text	BLOOMBERG

Child element	Schema type	Purpose	Example value
asset-class choice	—	Exactly one of <code><Equity></code> , <code><Bond></code> , <code><ShareClass></code> , <code><Warrant></code> , <code><Certificate></code> , <code><Option></code> , <code><Future></code> , <code><FXForward></code> , <code><Swap></code> , <code><Repo></code> , <code><FixedTimeDeposit></code> , <code><CallMoney></code> , <code><Account></code> , <code><Fee></code> , <code><RealEstate></code> , <code><REIT></code> , <code><Loan></code> , <code><Right></code> , <code><Commodity></code> , <code><PrivateEquity></code> or <code><CommercialPaper></code> — mandatory, exactly one	see Part III
<code><CapitalYieldsTaxClaim></code>	container	Tax-accrual details at the position level — rarely populated for non-German funds	—
<code><InflationaryAdjustment></code>	container	Inflation-linked adjustment fields for index-linked instruments	—
<code><RiskCodes></code>	list	Risk indicator values attached to this individual position	—
<code><Underlyings></code>	list	Underlying-instrument references for derivatives and structured products	—
<code><CustomAttributes></code>	Attribute sType	Producer-specific extensions	—

All children except `<UniqueID>`, `<TotalValue>`, and the asset-class choice are optional. The three mandatory members carry all the work of position identity, valuation, and asset-class branching, and everything above and below them in the table is supporting detail that producers populate when it matters.

Note on vocabulary: there is **no** `<Quantity>` element anywhere on `<Position>` and no `<PositionType>` element distinguishing long from short. The schema element named `<Position>` is of schema type `PositionType` — the latter being the name of the complex type in the XSD, not a tag name that ever appears in a FundsXML document. The word "quantity" in the prose that follows always refers to the generic concept of "how many of the instrument does the fund hold"; the actual XML tag depends on the asset class and takes its name from the specific choice child (`<Units>`, `<Nominal>`, `<Shares>`, `<Contracts>`).

Four points about the table deserve to be emphasised in plain text rather than left implicit.

`<TotalValue>` is mandatory on every position and is the one figure that reconciles against the fund total net assets from Chapter 5. A `<Position>` element with no `<TotalValue>` is a schema-validation failure; a `<Position>` whose `<TotalValue>` does not match the fund's `<TotalNetAssetValue>` when summed

across all positions is a business-validation failure. The consolidated total lives on the position, not on the asset-class-specific child, because the producer is the authoritative source for how the position rolls up into the fund currency and any downstream consumer should accept that roll-up without recomputing.

The asset-class-specific child is mandatory and the schema offers twenty-one variants. Every `<Position>` must end with exactly one of `<Equity>`, `<Bond>`, `<ShareClass>`, `<Warrant>`, `<Certificate>`, `<Option>`, `<Future>`, `<FXForward>`, `<Swap>`, `<Repo>`, `<FixedTimeDeposit>`, `<CallMoney>`, `<Account>`, `<Fee>`, `<RealEstate>`, `<REIT>`, `<Loan>`, `<Right>`, `<Commodity>`, `<PrivateEquity>` or `<CommercialPaper>`. This is where the per-class unit count, the per-unit price, and the detail fields live — *not* at the top of the position. An equity's `<Units>` field, for example, is a child of `<Equity>`, not of `<Position>`. A bond's `<Nominal>` field is a child of `<Bond>`. A cash account's balance lives in `<Account>/<MarketValue>`. The rule is strict and it organises everything that follows in Part III. (Producers describing an index or a crypto asset populate `AssetType=IX` or `AssetType=CR` on the asset-master side, but no corresponding choice exists on the position side; a fund holding such instruments emits them as `<Equity>` or `<CommercialPaper>` positions, depending on the operational convention.)

The concept of "quantity" does not correspond to a single schema element. Different asset classes describe their unit count with different tag names, each living inside the asset-class choice child:

- For an equity, `<Equity>/<Units>` carries the number of shares held.
- For a bond, `<Bond>/<Nominal>` carries the nominal value in the issue currency. A `<Nominal>1000000</Nominal>` means one million euros face value of bonds, which might be a single bond certificate or a thousand of them depending on the issue's denomination.
- For a fund held as a position, `<ShareClass>/<Shares>` carries the number of fund units.
- For an option or a future, `<Option>/<Contracts>` (respectively `<Future>/<Contracts>`) carries the number of contracts, and the value may be negative to indicate a short position — **FundsXML has no separate long/short flag**. The same applies to `<Swap>` legs.
- For cash accounts there is no unit count at all, because an account balance is already a total value carried in `<Account>/<MarketValue>`.

Section 6.7 will return to the bond case, which is the one that most frequently trips up implementers. The rule to remember is that each asset class has its own idiom and the element name is chosen to reflect it; **no element called `<Quantity>` exists anywhere in the schema**, and a producer writing `<Quantity>` will fail schema validation.

`<Amount>` elements always carry a `ccy` attribute. The FundsXML `<Amount>` element — whether inside `<TotalValue>`, `<Price>`, `<MarketValue>`, `<AveragePurchasePrice>`, or any other amount-bearing field — names its currency explicitly. In addition, optional boolean attributes `isFundCcy`, `isSubfundCcy`, and `isShareClassCcy` let the producer flag which amount is the fund-base, sub-fund, or share-class currency when several are present. A position holding Swiss Nestlé shares may therefore carry its `<TotalValue>` with two `<Amount>` entries — one `ccy="EUR" isFundCcy="true"` for the fund-base figure, and one `ccy="CHF"` for the native currency figure — both describing the same economic value at different FX rates. A consumer that ignores the `ccy` attribute and assumes "everything in the fund's base currency" will misread the Swiss position's native amount and the error will propagate into any downstream calculation.

One final remark before moving to the asset side. Every asset-class-specific field (coupon rate on a bond, strike on an option, counterparty LEI on a cash position) lives **inside the asset-class child**, not as a direct sibling of `<UniqueID>` or `<TotalValue>`. A bond position has `<TotalValue>` and a `<Bond>` sub-block containing

`<Nominal>`, `<Price>`, `<DirtyPrice>`, `<MarketValue>`, `<InterestClaimGross>`, `<AccruedInterestDays>`, and so on. A cash account position has `<TotalValue>` and an `<Account>` sub-block containing `<MarketValue>` and `<Interests>`. The asset-class sections in Part III walk through each sub-block in detail; Table 6.1 is the contract for everything *outside* them.

6.5 Asset-Level Fields in AssetMasterData

Symmetrical to §6.4, but from the other side of the linkage: the fields that appear on every `<Asset>` entry in `<AssetMasterData>` regardless of asset class. The division of labour between position and asset is straightforward: **the position answers the question "how much?", the asset answers the question "what is it?"**

Table 6.2 lists the common asset-level fields and their schema types, in the exact order in which they must appear inside `<Asset>`.

Table 6.2 — Common asset-level fields (schema type `AssetType`)

Field	Schema type	Mandatory	Purpose
UniqueID	xs:ID	yes	Globally unique identifier within the delivery; target of Position/UniqueID
Identifiers	IdentifiersType	no	ISIN, LEI, GermanWKN, SwissValorenCode, CUSIP, SEDOL, Bloomberg, ReutersRIC, OtherID
DataSupplier	DataSupplierType	no	Producer of this asset record (usually inherited from ControlData)
Currency	ISOCurrencyCodeType	yes	Nominal currency of the instrument (three-letter ISO 4217 code)
IncomeCurrency	ISOCurrencyCodeType	no	Currency in which the instrument's income is paid, if different
Country	ISOCountryCodeType	no	Country of issue (<i>Emissionsland</i>)
Name	Text256Type	yes	Human-readable instrument name
FISN	xs:string (128)	no	ISO 18774 Financial Instrument Short Name
AssetType	enum (2 chars)	yes	Discriminator: EQ / B0 / SC / WA / CE / OP / FU / FX / SW / RP / FT / CM / AC / FE / RE / RT / L0 / RI / C0 / PE / CP / IX / CR
AssetDetails	AssetDetailsType	no	A choice of 23 inline types carrying the asset-class-specific fields
CountrySpecificData	container	no	National reporting extensions (AT, DE, DK, FR, LU, NL)
Securitized	xs:boolean	no	Whether the asset is securitised
IsInfrastructureInvestment	xs:boolean	no	Infrastructure investment flag
InfrastructureInvestmentType	enum	no	Infrastructure category
Ratings	RatingsType	no	Credit ratings from agencies (one entry per agency)
RealEstateCompany	CompanyType	no	Corresponding real-estate company (for direct property holdings)

Field	Schema type	Mandatory	Purpose
Classifications	ClassificationType	no	Sector/industry/style classifications, same structure as in Chapter 5
CustomAttributes	AttributesType	no	Producer-specific extensions

Four points about the table deserve to be emphasised in plain text.

The `AssetType` element is a two-character code, not an English word. The values are `EQ` for equity, `B0` for bond, `SC` for fund share class, `WA` for warrant, `CE` for certificate, `OP` for option, `FU` for future, `FX` for FX forward, `SW` for swap, `RP` for repo, `FT` for fixed-time deposit, `CM` for call money, `AC` for account, `FE` for fee, `RE` for real estate, `RT` for REIT, `LO` for loan, `RI` for right, `CO` for commodity, `PE` for private equity, `CP` for commercial paper, `IX` for index, `CR` for crypto. These are the operational codes used by fund accounting systems across Europe, and producers coming from such systems usually have the mapping built in already. A consumer that reads `AssetType` branches its parser on the two-letter code, loads the corresponding `AssetDetails` child, and applies the asset-class-specific logic. There is no enumeration with English labels anywhere in the schema.

The ISIN is not a direct child of `<Asset>`. The ISIN lives inside `<Identifiers>`, the same container we met at the fund and share-class level in Chapter 5. So does the LEI (used for issuers, not for the asset itself), the GermanWKN, the SwissValorenCode, and every other instrument code. A typical equity asset entry begins with `<UniqueID>`, then `<Identifiers><ISIN>...</ISIN><GermanWKN>...</GermanWKN></Identifiers>`, then `<Currency>`, then `<Country>`, then `<Name>`, then `<AssetType>`, then `<AssetDetails>`. Producers that write the ISIN as a top-level child will fail schema validation.

`AssetDetails` is the home of asset-class-specific fields, not a collection of siblings of `<Asset>`. Everything that distinguishes an equity from a bond — the issuer block, the stock market, the coupon, the maturity, the strike, the counterparty of a forward, the counterparty of a cash account — lives one level below, inside an `<AssetDetails>` child whose content is exactly one of twenty-three sub-elements (`Equity`, `Bond`, `ShareClass`, `Warrant`, `Certificate`, `Option`, `Future`, `FXForward`, `Swap`, `Repo`, `FixedTimeDeposit`, `CallMoney`, `Account`, `Fee`, `RealEstate`, `REIT`, `Loan`, `Right`, `Commodity`, `PrivateEquity`, `CommercialPaper`, `Index`, `Crypto`). The element names match the ones on the position-side choice of §6.4 deliberately, so that the discriminator on both sides is the same word, not a mapping a consumer has to learn.

Issuer and instrument are distinct, and the schema respects that distinction. An asset entry has two identity layers. The *instrument* is the tradable unit — a specific equity share class, a specific bond issue, a specific derivative contract — and it is identified at the top of `<Asset>` by its own `<Identifiers>` block, whose `<ISIN>` (or `<UniqueID>`, for instruments without an ISIN) is the primary key. The *issuer* is the legal entity that issued the instrument — and it is identified by a separate `<Issuer>` block of schema type `CompanyType` that lives **inside** `AssetDetails/<Equity>` or `AssetDetails/<Bond>`. `CompanyType` carries its own `<Identifiers>` container with the issuer's LEI, a mandatory `<Name>`, optional `<LegalName>`, `<LegalForm>`, `<Address>`, `<BusinessCountry>`, `<ParentCompany>` for corporate hierarchies, an `<OtherClassification>` container, and a `<Type>` enumeration covering the ECB sector categories (`Non-financial corporations`, `Central bank`, `Deposit-taking corporations except the central bank`, `General government`, `Insurance corporations`, `Pension funds`, `Money market funds`, `Non-MMF investment`

funds , and five others). For equities the instrument-issuer relationship is typically one-to-one: the Siemens AG legal entity has one LEI, and DE0007236101 is the ordinary common stock of that entity. For bonds the relationship is one-issuer-to-many-instruments: the Federal Republic of Germany has one LEI but hundreds of bund series, each with its own ISIN, all backed by the same `<Issuer>` block carrying the Federal Republic's LEI and `Type="General government"` . Section 6.7 returns to this point in the context of bond-specific fields.

All asset-class-specific fields — the strike of an option, the maturity of a bond, the counterparty of a forward — live inside `<AssetDetails>` below the common fields of Table 6.2. The five sections that follow treat them one asset class at a time.

6.6 Equities

The primary case for the Europa Growth Fund. Three subsections.

6.6.1 The Equity Asset Model

An equity asset entry in `AssetMasterData` describes a tradable share class of a public company. Its structure has two layers: the generic asset-level fields of Table 6.2, populated with `AssetType=EQ` , followed by an `AssetDetails/Equity` child whose content follows the schema type `EquityType` . The generic fields carry identity and classification; the equity-specific fields carry listing, issuer, and equity-specific metadata.

Identifiers live inside the top-level `<Identifiers>` block of Table 6.2, not inside `<Equity>` . The primary identifier is the ISIN, twelve characters, ISO 6166. For securities that also trade in non-ISIN-using markets, FundsXML carries additional identifier fields in the same container: `SEDOL` for UK listings, `CUSIP` for North-American instruments, and `GermanWKN` for German securities. Bloomberg tickers and Reuters RICs have their own dedicated fields (`Bloomberg` with `Ticker` , `Exchange` , `Market` , `BBGID` sub-children; `ReutersRIC` with a `type` attribute).

Inside `<Equity>` , the schema type `EquityType` (XSD §25059) carries the following children in order:

- `ClassOfBusiness` — a structured `IndustryProviderType` block with `Offeror` , `Date` , `Value` , and optional `Text` , naming the sector classification provider (e.g. *Bloomberg*, *GICS*) and the value.
- `StockMarket` — one or more `MICCodeType` values (ISO 10383 four-character market identifier codes) naming the exchanges on which the equity is listed. `XETR` for Xetra, `XPAR` for Euronext Paris, `XVTX` for SIX Swiss Exchange, `XAMS` for Euronext Amsterdam, `XLON` for the London Stock Exchange, `XMAD` for Bolsa de Madrid. The element repeats for multi-listed stocks.
- `Issuer` — a structured `CompanyType` block identifying the issuing legal entity: `<Identifiers>` with the issuer's LEI, a mandatory `<Name>` , an optional `<LegalForm>` , an optional `<BusinessCountry>` , and an optional `<Type>` from the ECB sector enumeration.
- `Listing` — an enumeration `A / B / G / K / M / S / V / Y / Z` indicating the kind of exchange listing. `A` (official market) is the ordinary case for blue-chip European stocks; `K` means *not listed*, used for private or unlisted issues.

- **ParValue** — the nominal value of the share in the instrument's currency (often 1.00 for modern no-par-value shares).
- **WithholdingTaxRate** — the producer's default withholding tax rate on dividends from this instrument.
- **MarketCapitalization** — a **Date / Value** pair carrying the full market capitalisation of the issuer as at that date.
- **CustomMarketCapitalization** — a **Name / Date / Value** triple where **Name** is drawn from the enumeration **Small Cap**, **Small & Mid Cap**, **Mid Cap**, **Mid & Large Cap**, **Large Cap**, **All Cap**, **Other**. This is the field a factsheet engine reads when it needs a size bucket.
- **Industries** — a container for one or more **IndustryCode** entries, each with **Name** (the classification scheme, e.g. **GICS**), **Date**, and **Value** (the numeric or textual code within that scheme).

Classification through **AssetType/Classifications**. The generic-asset **<Classifications>** block from Table 6.2 is the more idiomatic place to carry sector-classification values for downstream consumers. Following the same pattern as in Chapter 5, each **<Classification>** entry names a provider via **<ListedGroup>** (**EFAMA**, **MORNINGSTAR**, **LIPPER**, **BLOOMBERG**, **MIFID**, **BVI**, **VOEIG**, **ESMA**, **AMF**, **WM**, **CIC**, **CFI**) or **<UnlistedGroup>** (free text, used for **GICS** and **ICB** which are not in the enumeration) and supplies one or more **<Value>** elements with optional **level** attributes for hierarchies. The **<Industries>** and **<ClassOfBusiness>** fields inside **<Equity>** are historical and mostly populated by legacy fund-accounting systems; modern producers prefer **<Classifications>**.

Table 6.3 — Equity-specific fields inside **AssetDetails/Equity (schema type **EquityType**)**

Field	Schema type	Example	Mandatory
ClassOfBusiness	IndustryProviderType	Offeror= <i>Bloomberg</i> , Value= <i>BICS</i>	optional
StockMarket	MICCodeType (repeats)	XETR	optional
Issuer	CompanyType	see text above	optional
Listing	enum A / B / G / K / M / S / V / Y / Z	A	optional
ParValue	xs:decimal	1.00	optional
WithholdingTaxRate	xs:decimal	26.375	optional
MarketCapitalization	Date + Value	2026-03-31, 145 bn	optional
CustomMarketCapitalization	Name enum + Date + Value	Large Cap	optional
Industries	list of IndustryCode	GICS 20106010	optional

All equity-level fields are optional; an `<Equity>` element with no children at all is schema-valid (an empty but schema-valid element). In practice, producers populate `Issuer`, `StockMarket`, and at least one classification entry for every equity in the portfolio; the rest depend on the producer's downstream consumers.

6.6.2 Position-Level Fields Specific to Equities

At the position level, an equity entry ends with an `<Equity>` child (defined inline as an anonymous complex type within `<Position>`, XSD §28571) whose mandatory first child is `<Units>`. Its sequence is:

- `<Units>` (mandatory, `xs:decimal`) — the number of shares the fund holds. Almost always a whole number for traditional long-only UCITS; fractional shares exist in a few specialised vehicles (robo-advisors offering fractional investment, some unit-linked wrappers) but do not appear in the Europa Growth Fund or in most European institutional portfolios.
- `<Price>` (optional, `FundAmountType`) — the last available quote, in the listing currency, typically the closing price on the valuation date at the primary listing. If the stock was suspended for the day, the price holds at the last valid quote and the position's `<PriceDate>` (inside the parent `<Position>`) reflects the age of the quote.
- `<OtherPrices>` (optional) — additional prices (hold-to-maturity-style alternatives, rarely used for equities).
- `<MarketValue>` (optional, `FundAmountType`) — `Units × Price`, converted into the fund's base currency if the listing currency differs. The conversion rate is the producer's reference rate at the valuation point.
- `<BookRate>` (optional, `FundAmountType`) — the book value used for capital-gains accounting.
- `<PurchaseValue>` (optional, `FundAmountType`) — the historical cost of the position.
- `<DividendsDue>` (optional, `FundAmountType`) — the dividend income that has accrued between the last ex-dividend date and the valuation date but has not yet been paid out. Most producers do not populate it; they treat dividends as discrete events that flow through `Distributions` on the share class (Chapter 7) rather than accruing into the equity position.
- `<LendedFlag>` (optional, `xs:boolean`) — whether part of the position is out on loan to a counterparty.
- `<LendedUnits>` (optional, `xs:decimal`) — the number of shares out on loan.
- `<IndicatorRaffled>` (optional, `xs:boolean`) — whether the security has been drawn by lot in a corporate action.

Note what is **not** a child of `<Equity>` but lives one level up on `<Position>`: `TotalValue`, `TotalPercentage`, `Currency`, `PriceDate`, `PricingSource`. Those are position-level concerns and are common to every asset class. A producer that writes `<TotalValue>` as a sibling of `<Units>` inside `<Equity>` is wrong; the field must appear at `<Position>` level.

6.6.3 Equity Example: A Siemens Position

A complete pair — position and asset entry — for the Europa Growth Fund's Siemens holding on 31 March 2026.

```
<!-- Inside Fund/FundDynamicData/Portfolios/Portfolio/Positions -->
<Position>
  <UniqueID>DE0007236101</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
```

```

    <Amount ccy="EUR" isFundCcy="true">8125000.00</Amount>
  </TotalValue>
  <TotalPercentage>1.75</TotalPercentage>
  <PriceDate>2026-03-31</PriceDate>
  <Equity>
    <Units>50000</Units>
    <Price>
      <Amount ccy="EUR">162.50</Amount>
    </Price>
    <MarketValue>
      <Amount ccy="EUR" isFundCcy="true">8125000.00</Amount>
    </MarketValue>
  </Equity>
</Position>

<!-- Elsewhere, inside the root AssetMasterData block -->
<Asset>
  <UniqueID>DE0007236101</UniqueID>
  <Identifiers>
    <ISIN>DE0007236101</ISIN>
    <GermanWKN>723610</GermanWKN>
  </Identifiers>
  <Currency>EUR</Currency>
  <Country>DE</Country>
  <Name>Siemens AG</Name>
  <AssetType>EQ</AssetType>
  <AssetDetails>
    <Equity>
      <StockMarket>XETR</StockMarket>
      <Issuer>
        <Identifiers>
          <LEI>W38RGI023J3WT1HWRP32</LEI>
        </Identifiers>
        <Name>Siemens Aktiengesellschaft</Name>
        <BusinessCountry>DE</BusinessCountry>
        <Type>Non-financial corporations</Type>
      </Issuer>
      <Listing>A</Listing>
      <CustomMarketCapitalization>
        <Name>Large Cap</Name>
        <Date>2026-03-31</Date>
        <Value>145000000000</Value>
      </CustomMarketCapitalization>
      <Industries>
        <IndustryCode>
          <Name>GICS</Name>
          <Date>2026-03-31</Date>
          <Value>20106010</Value>
        </IndustryCode>
      </Industries>
    </Equity>
  </AssetDetails>

```

```

<Classifications>
  <Classification>
    <UnlistedGroup>GICS</UnlistedGroup>
    <Type>Sector</Type>
    <Language>en</Language>
    <Value level="1">Industrials</Value>
    <Value level="2">Industrial Conglomerates</Value>
  </Classification>
</Classifications>
</Asset>

```

Reading field by field: the position's `<UniqueID>` is the ISIN, following the convention of §6.3.2 (and satisfying the `xs:ID` constraint because `DE0007236101` begins with a letter). The consolidated `<TotalValue>` is 8.125 million euros — approximately 1.75% of the fund's total net assets from Chapter 5 — and lives in the fund's base currency, flagged `isFundCcy="true"`. Inside `<Equity>`, the position says the fund holds 50,000 Siemens shares, last priced at 162.50 EUR at Xetra's close on 31 March 2026, with a market value equal to `Units × Price` also in euros.

The asset entry confirms the instrument identity. `<Identifiers>` carries the ISIN and the six-character German WKN (723610, the historical code that German retail systems still use alongside the ISIN). The top-level `<Currency>` names the instrument's nominal currency (EUR) and `<Country>` names the country of issue (DE). `<AssetType>EQ</AssetType>` signals the asset-class to consumers. Inside `<AssetDetails>/<Equity>`, the listing information — Xetra (XETR) as the stock market, A (official market) as the listing category — sits above the issuer block. The issuer block is a full `<Issuer>` element of schema type `CompanyType`, with Siemens AG's LEI (20 characters, ISO 17442), a mandatory name, a business country, and an ECB-sector classification of `Non-financial corporations`. A `<CustomMarketCapitalization>` entry places Siemens in the `Large Cap` bucket as at 31 March 2026 with a rounded-EUR figure, and an `<Industries>` entry carries the GICS industry code 20106010 (Industrial Conglomerates) for downstream consumers that speak that taxonomy.

The top-level `<Classifications>` block — sibling of `<AssetDetails>` — duplicates the GICS classification in the idiomatic form that the rest of the book uses: an `<UnlistedGroup>GICS</UnlistedGroup>` (because `GICS` is not a listed provider in the schema's enumeration), a `<Type>Sector</Type>`, a `<Language>en</Language>`, and two `<Value>` elements with `level` attributes walking the hierarchy from Industrials down to Industrial Conglomerates. A consumer that reads this block picks whichever scheme it recognises.

A consumer that produces a factsheet for the fund renders Siemens as "Siemens AG, 1.75%, Industrials, Germany" using five or six of the visible fields. A consumer that produces a regulatory report reads the LEI and the GICS code for their respective disclosures. The same pair serves every downstream use case without any additional data.

This pair is the template that §6.12 will repeat fifteen times with different companies, sectors, and countries. The fields are always the same; the values vary.

6.7 Bonds

The Europa Growth Fund holds no bonds, but any treatment of FundsXML portfolios must cover them because bonds are structurally the most different asset class from equities and because most European fund portfolios outside pure equity mandates contain at least some. The example in this section comes from a generic European bond fund rather than from the Europa Growth Fund.

Bonds are more complex than equities for one overriding reason: a bond carries two identities rather than one. The *instrument* is a specific bond issue with a specific maturity and coupon; the *issuer* is the legal entity that issued it. Where an equity share class is in one-to-one correspondence with its issuer's LEI, a single issuer may have issued dozens or hundreds of distinct bonds, each with its own ISIN and its own behaviour, all backed by the same credit. The bond asset model must represent both layers.

6.7.1 The Bond Asset Model

An `<Asset>` entry for a bond carries `AssetType=B0`, and its bond-specific details live inside `<AssetDetails>/<Bond>` (schema type `BondType`, XSD §2012). The type is large — thirty or more optional children — and the sequence below lists only the fields that matter for a typical European fund portfolio.

Convertible flag. The one *mandatory* child of `BondType` is `<ConvertibleFlag>` (`xs:boolean`). Every bond must state whether it carries equity conversion rights, even if the answer is trivially *false* for a straight government bond. Convertible bonds have an additional `<ConvertibleBond>` sub-block with strike and conversion ratio details.

Issuer. A structured `CompanyType` block identifying the issuing legal entity — the same type we saw for equities in §6.6.1, but the issuer for a bond is almost always populated more richly because rating agencies, regulatory reporting, and concentration limits all hang on it. `Issuer/Identifiers/LEI` is the single most important field on the whole entry.

Listing. `<StockMarket>` (repeatable MIC codes) and `<Listing>` (the same A/B/G/K/M/S/V/Y/Z enumeration as for equities) identify the exchanges on which the bond trades. A `<ListingUnit>` field, unique to bonds, distinguishes `P` (pieces — bonds denominated in whole certificates) from `N` (nominal amount — the more common case in Europe). For a typical German bund, `ListingUnit=N`.

Maturity and key dates.

- `IssueDate` — the date the bond was originally issued.
- `MaturityDate` — the date on which the principal is repaid to investors.
- `DateFirstCoupon` and `DateLastCoupon` — the boundary coupon dates.
- `CouponDate` — the next scheduled coupon payment.

Coupon. The cash flow the bond pays to investors during its life is carried inside a `<Coupon>` sub-block with four children:

- `<Type>` — an enumeration `fix / float / zero / others`. Note the lowercase spelling; the schema uses `fix`, not `Fixed`.

- `<PaymentFrequency>` (`FrequencyType`) — a SWIFT-style four-letter code: `DAIL` (daily), `WEEK` (weekly), `TWVK` (two-weekly), `MNTH` (monthly), `TWMN` (two-monthly), `QURT` (quarterly), `SEMI` (semi-annual), `YEAR` (annual). Most European government bonds pay `YEAR`, most corporate bonds `SEMI`. These are the same codes the EMT/EPT regulatory templates of Chapter 8 use, so producers coming from that side of the house have the mapping already.
- `<InterestRate>` (`xs:decimal`) — the nominal coupon rate as a percentage of nominal value (`2.50` for a 2.50% coupon).
- For floating-rate notes, `<BaseIndex>` (`IdentifiersType`) and `<OffsetBPS>` (`xs:decimal`) describe the floating rate formula.

Redemption. A `<RedemptionRate>` (`xs:decimal`) carries the price at which the bond will be redeemed at maturity — typically `100.00` (par). An optional `<Redemption>` sub-block carries the redemption type (`Bullet` or `Sinkable`), any call/put option date, the direction of the option (`Issuer`, `Bearer`, or `Both`), and the option strike price where relevant.

Tax fields. `<WithholdingTaxRate>`, `<EUWithholdingTaxRate>`, `<EUWithholdingTaxCategory>` (`A / B / C`) and `<CapitalYieldsTaxKind>` cover the various European withholding regimes. Most producers populate only the first.

The single biggest surprise in the bond type. There is *no* `BondType` enumeration. The schema does not distinguish a *Government* bond from a *Corporate* bond from a *Covered* bond at the `<Bond>` level; that distinction is carried through `Issuer/Type` (the ECB-sector enumeration: `General government`, `Non-financial corporations`, `Deposit-taking corporations`, and so on) and through the `<Classifications>` block on `<Asset>`. Similarly, there is *no* `CountryOfRisk` element, *no* `DayCountConvention` enumeration, and *no* `Seniority` field. The producer who needs these concepts must carry them in `<Classifications>` or `<CustomAttributes>`.

Credit ratings live on `<Asset>` itself (not inside `<Bond>`) under `<Ratings>`, schema type `RatingsType`. The structure is slightly unusual: the outer container has one `<Rating>` child per *agency*, and each agency `<Rating>` wraps a `RatingCompany` name and one or more inner `<Rating>` entries carrying the actual rating values. An agency's rating history is a sequence of inner entries; the current rating is the most recent one. A bond rated by three agencies therefore has three outer `<Rating>` entries, one per agency, each with at least one inner `<Rating>` entry.

Table 6.4 — Key bond-specific fields inside `AssetDetails/Bond` (schema type `BondType`)

Field	Schema type	Example	Mandatory
ConvertibleFlag	xs:boolean	false	yes
Issuer	CompanyType	Federal Republic of Germany	optional but expected
StockMarket	MICCodeType (repeats)	XETR	optional
Listing	enum A / B / G / K / M / S / V / Y / Z	A	optional
ListingUnit	enum P / N	N	optional
IssueDate	xs:date	2024-02-15	optional
MaturityDate	xs:date	2034-02-15	optional
CouponDate	xs:date	2027-02-15	optional
Coupon/Type	enum fix / float / zero / others	fix	optional
Coupon/PaymentFrequency	FrequencyType	YEAR	optional
Coupon/InterestRate	xs:decimal	2.50	optional
RedemptionRate	xs:decimal	100.00	optional
Redemption/Type	enum Bullet / Sinkable	Bullet	optional
InterestRate	xs:decimal	2.50	optional
InterestsStartDate	xs:date	2026-02-15	optional

6.7.2 Position-Level Fields Specific to Bonds

At the position level, a bond entry ends with a `<Bond>` child (defined inline as an anonymous complex type within `<Position>`, XSD §28640) whose mandatory first child is `<Nominal>`. Its sequence is:

- `<Nominal>` (mandatory, `xs:decimal`) — the nominal value of the bonds held, in the issue currency.
`<Nominal>1000000</Nominal>` means one million of nominal value, not one million pieces. This is the single most confusing convention in bond accounting for first-time implementers, and it is worth stating three times: bond `<Nominal>` is nominal, bond `<Nominal>` is nominal, bond `<Nominal>` is nominal. The total value of a bond position at par is therefore $\text{Nominal} \times (\text{Price} / 100)$, because `Price` is expressed as a percentage of nominal.
- `<OriginalFace>` (optional) — the par value at the time of purchase, used for sinking-fund bonds whose nominal has been reduced through partial redemptions.

- `<Price>` (optional, `FundAmountType`) — the **clean price**, expressed as a percentage of nominal value. A bond trading at par has price 100.00, a bond at a five-point discount has price 95.00.
- `<OtherPrices>` (optional) — additional price quotes (hold-to-maturity, for example).
- `<DirtyPrice>` (optional, `FundAmountType`) — the clean price plus accrued interest, expressed as a percentage of nominal.
- `<MarketValue>` (optional, `FundAmountType`) — the market value of the position *without interest*, computed as $\text{Nominal} \times (\text{Price} / 100)$.
- `<InterestClaimNet>` and `<InterestClaimGross>` (optional, `FundAmountType`) — the net and gross accrued interest in the bond's currency. The net/gross distinction reflects withholding tax treatment.
- `<AccruedInterestDays>` (optional, `xs:integer`) — the number of days of interest that have accrued, as an audit field for consumers that want to verify the producer's computation.
- `<ZeroBondInterestClaimNet>`, `<ZeroBondInterestClaimGross>`, `<ZeroBondCapitalYieldsTaxClaim>`, `<ZeroBondYield>` — analogues for zero-coupon bonds.
- `<Indexfactor>`, `<Poolfactor>` — for index-linked and pass-through bonds.
- `<BookRate>`, `<PurchaseValue>` — cost basis fields.
- `<LendedFlag>`, `<LendedUnits>`, `<IndicatorRaffled>`, `<CollateralValue>` — loan and corporate-action fields shared with equities.

Clean price, dirty price, and position total. Bonds trade on clean price in European markets — a price that excludes accrued interest. The clean price is what gets quoted on trading screens and in newspaper tables. When an investor actually buys a bond, they pay the clean price plus the accrued interest since the last coupon date — the *dirty price* — because the seller is entitled to the portion of the next coupon that has accumulated during their ownership. FundsXML follows the clean-price convention for `<Bond>/<Price>` and models the accrued interest through `<Bond>/<InterestClaimGross>` (or `InterestClaimNet`) as a separate field, and optionally through `<Bond>/<DirtyPrice>` as the combined figure.

The position's consolidated `<TotalValue>` at the `<Position>` level is therefore:

$$\text{TotalValue} = \text{Nominal} \times (\text{Price} / 100) + \text{InterestClaimGross}$$

A consumer that computes the total as $\text{Nominal} \times \text{Price}$ without the percentage scaling will get an answer that is wrong by a factor of one hundred. A consumer that reads only `<Bond>/<MarketValue>` and omits the accrued interest will understate the position by the coupon accrual — small in absolute terms but meaningful for yield and duration calculations. In practice, the producer emits all three fields (`MarketValue`, `InterestClaimGross`, and the parent `<Position>/<TotalValue>` as the sum), and the consumer reads whichever it needs.

Optional analytics such as yield-to-maturity, modified duration, and convexity are not first-class fields of `<Bond>`. A producer that wants to ship them does so either through `<OtherPrices>` with a descriptive type label or through `<CustomAttributes>` on the asset entry. Most producers do not populate them; consumers who need them compute them locally from the bond's characteristics.

6.7.3 Bond Example: A German Bund Position

A generic European bond fund holds a position in a ten-year German federal government bond: nominal value 1,000,000 EUR, coupon 2.50%, maturity 15 February 2034, clean price 98.75 at the 31 March 2026 valuation. The accrued interest since the last coupon payment on 15 February 2026 is approximately 3,150 EUR (44 days at 2.50% on 1,000,000 EUR nominal).

```
<!-- Portfolio position -->
<Position>
  <UniqueID>DE0001102614</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">990650.68</Amount>
  </TotalValue>
  <TotalPercentage>0.99</TotalPercentage>
  <PriceDate>2026-03-31</PriceDate>
  <Bond>
    <Nominal>1000000</Nominal>
    <Price>
      <Amount ccy="EUR">98.75</Amount>
    </Price>
    <DirtyPrice>
      <Amount ccy="EUR">99.06507</Amount>
    </DirtyPrice>
    <MarketValue>
      <Amount ccy="EUR" isFundCcy="true">987500.00</Amount>
    </MarketValue>
    <InterestClaimGross>
      <Amount ccy="EUR">3150.68</Amount>
    </InterestClaimGross>
    <AccruedInterestDays>44</AccruedInterestDays>
  </Bond>
</Position>

<!-- Corresponding asset entry -->
<Asset>
  <UniqueID>DE0001102614</UniqueID>
  <Identifiers>
    <ISIN>DE0001102614</ISIN>
  </Identifiers>
  <Currency>EUR</Currency>
  <Country>DE</Country>
  <Name>Bundesrepublik Deutschland 2.50% 02/2034</Name>
  <AssetType>B0</AssetType>
  <AssetDetails>
    <Bond>
      <ConvertibleFlag>false</ConvertibleFlag>
      <Issuer>
        <Identifiers>
          <LEI>529900PY3M0X01E5MD14</LEI>
        </Identifiers>
      </Issuer>
    </Bond>
  </AssetDetails>
</Asset>
```



```

    <Name>Federal Republic of Germany – Finance Agency</Name>
    <BusinessCountry>DE</BusinessCountry>
    <Type>General government</Type>
  </Issuer>
  <ListingUnit>N</ListingUnit>
  <IssueDate>2024-02-15</IssueDate>
  <MaturityDate>2034-02-15</MaturityDate>
  <CouponDate>2027-02-15</CouponDate>
  <Coupon>
    <Type>fix</Type>
    <PaymentFrequency>YEAR</PaymentFrequency>
    <InterestRate>2.50</InterestRate>
  </Coupon>
  <RedemptionRate>100.00</RedemptionRate>
  <Redemption>
    <Type>Bullet</Type>
  </Redemption>
  <InterestRate>2.50</InterestRate>
  <InterestsStartDate>2026-02-15</InterestsStartDate>
</Bond>
</AssetDetails>
<Ratings>
  <Rating>
    <RatingCompany>MOODY'S</RatingCompany>
    <Rating>
      <Date>2025-06-15</Date>
      <Value>Aaa</Value>
      <Description>Long-term issuer credit rating – stable outlook</Description>
    </Rating>
  </Rating>
  <Rating>
    <RatingCompany>STANDARD - POORS</RatingCompany>
    <Rating>
      <Date>2025-05-20</Date>
      <Value>AAA</Value>
      <Description>Long-term foreign currency – stable outlook</Description>
    </Rating>
  </Rating>
  <Rating>
    <RatingCompany>FITCH</RatingCompany>
    <Rating>
      <Date>2025-07-02</Date>
      <Value>AAA</Value>
      <Description>Long-term issuer default rating – stable outlook</Description>
    </Rating>
  </Rating>
</Ratings>
<Classifications>
  <Classification>
    <ListedGroup>ESMA</ListedGroup>
    <Type>Bond Type</Type>
    <Language>en</Language>
  </Classification>
</Classifications>

```

```

    <Value>Sovereign</Value>
  </Classification>
</Classifications>
</Asset>

```

Reading this pair: the position holds one million euros nominal of the 2.50% bund maturing 15 February 2034. The clean price is 98.75 (a small discount to par, reflecting a yield slightly above the coupon). The accrued interest of 3,150.68 EUR is the coupon income earned in the forty-four days since the 15 February 2026 payment, expressed as `<InterestClaimGross>` inside `<Bond>`. The position-level `<TotalValue>` is 990,650.68 EUR — the sum of `<Bond>/<MarketValue>` ($987,500.00 \text{ EUR} = 1,000,000 \times 0.9875$) and the interest claim. The `<DirtyPrice>` of 99.06507 is the clean price plus accrued expressed as a percentage, and consumers that prefer to read a single price number use it.

The asset entry carries the mandatory `<ConvertibleFlag>false</ConvertibleFlag>` as the first child of `<Bond>` — even for a vanilla government bond. The `<Issuer>` block names the Federal Republic's bond-issuing agency with its LEI, a business country of DE, and an ECB sector classification of General government; this is where the "sovereign" nature of the bond is encoded, not in any `BondType` enumeration. The `<ListingUnit>N</ListingUnit>` declares that the bond trades by nominal, matching the position-level `<Nominal>` idiom. The `<Coupon>` sub-block uses the `<PaymentFrequency>YEAR</PaymentFrequency>` code for an annual coupon, and `<Type>fix</Type>` (lowercase, as the schema demands) for the fixed-rate nature. The `<Redemption><Type>Bullet</Type></Redemption>` declares that the bond is redeemed in a single payment at maturity, not in a sinking-fund pattern.

The `<Ratings>` block carries three agency entries — Moody's, Standard & Poor's, and Fitch — each wrapping a `<RatingCompany>` name and one inner `<Rating>` entry with date, value, and description. All three agree on AAA (or Aaa in Moody's scale), as one would expect for German government paper. A consumer that needs the *middle* of the three ratings can read all three and apply its own logic; a consumer that wants the *most conservative* picks the lowest. A `<Classifications>` entry under `<ListedGroup>ESMA</ListedGroup>` classifies the bond as Sovereign for downstream consumers that want a textual label without walking into the issuer type.

The bond example shows every structural feature that distinguishes bonds from equities: the nominal-quantity convention, the clean/dirty price split, the rich issuer information, the multiple rating agencies, and the surprising absence of a `BondType` enumeration in favour of issuer-sector and classifications-based categorisation. A European bond fund's portfolio is a long list of entries of this shape, one per issue.

6.8 Investment Funds

The case in which a fund holds another fund's shares as a position. For a pure equity UCITS like the Europa Growth Fund this is a minor pattern — the fund occasionally holds a European equity ETF as a liquidity substitute — but for a fund-of-funds it is the dominant pattern, and either way the asset class deserves its own treatment.

6.8.1 The Fund-as-an-Asset Model

A held fund is represented in `AssetMasterData` as an `<Asset>` element whose `<AssetType>` is `SC` (*Share Class*), not a nested `<Fund>` element. The discriminator is a deliberate reminder that from the holder's perspective, what is owned is always a *specific share class* of a fund — the euro-distributing class, the CHF-hedged class, the institutional class — never the fund as an abstraction. In Chapter 5 we spent twenty pages establishing that investable units are share classes; here we see the same principle applied from the other side.

A fund-of-funds that holds six underlying funds therefore carries six `<Asset>` entries with `AssetType=SC`, not six nested `<Fund>` elements. Each position in the portfolio has a `<ShareClass>` child carrying its unit count and valuation, and each asset entry describes the held share class as an instrument. The portfolio-asset linkage mechanism is uniform across all asset classes, and introducing a nested-fund exception would complicate every consumer's parsing logic for no benefit.

Share-class details inside `AssetDetails/<ShareClass>` (schema type `ShareClassDetailsType`, XSD §32804) are surprisingly minimal. The schema treats a held share class as just another tradable instrument, and most of the rich fund-level metadata from Chapter 5 (legal structure, domicile, benchmarks, classifications, SFDR product type) is *not* duplicated on the holder's side. `ShareClassDetailsType` carries only:

- `Issuer` — a `CompanyType` block identifying the fund's management company or promoter.
- `StockMarket` — one or more MIC codes for the exchanges on which the share class trades (relevant only for listed share classes such as ETFs).
- `Listing` — the same A/B/G/K/M/S/V/Y/Z enumeration as for equities.
- `ListingUnit` — `U` (units) or `N` (nominal), almost always `U`.
- `IssueDate` — the share class's inception date.
- `MaturityDate` — the share class's maturity date, for closed-ended vehicles.
- `WithholdingTaxRate` — the tax rate applied to distributions.
- `ShareClassType` — a `ShareClassTypeType` block with `Code`, `EarningUse` (D/P/R), and short/long names, mirroring the structure we met at `ShareClass/ShareClassType` in Chapter 5.
- `ExchangeTradedFundFlag` (`xs:boolean`) — a dedicated flag for ETFs, distinct from the generic listing enumeration.

Fields that the schema does not carry on a held share class are, notably, `FundLegalType`, `FundCategory`, `OngoingCharges`, and any explicit *lookthrough* flag. The legal type of a held fund, where relevant, lives in `<Classifications>` on the outer `<Asset>` (for example `<ListedGroup>ESMA</ListedGroup>` with `<Value>UCITS</Value>`). The held fund's ongoing charges figure — relevant for the holding fund's own MiFID II cost disclosure — is not a field on the instrument at all; it is delivered through the regulatory templates of Chapter 8, specifically the EMT section that aggregates per-instrument charges, and through `<CustomAttributes>` on the asset entry for producers that want to carry it inline.

The *lookthrough* concept deserves a short explanation. For certain regulatory calculations — concentration limits, ESG exposure aggregation, Solvency II capital charges — the regulator asks not "what funds does this fund hold?" but "what underlying assets does this fund ultimately hold?" Lookthrough requires the holder to obtain the underlying composition of each held fund and aggregate it into its own exposure calculation. FundsXML carries the decomposition data itself through `Portfolio/PositionsDecomposed`, a dedicated sibling of `<Positions>` that replaces held share classes with the scaled-down positions of the underlying fund. A producer that wants to offer

lookthrough data to its consumers populates `<PositionsDecomposed>` with the underlying positions and a `<LookThroughLevels>` count stating how deep the decomposition went; a consumer that needs lookthrough reads from `<PositionsDecomposed>` rather than from `<Positions>`. There is no boolean "lookthrough required" flag anywhere on the share-class instrument.

6.8.2 Fund Position Example

A small illustrative example — not from the Europa Growth Fund, because a pure equity UCITS rarely holds ETFs of its own asset class, but from a generic European multi-asset fund that parks liquidity in the *iShares Core MSCI Europe UCITS ETF EUR (Acc)* (ISIN IE00B4K48X80) while awaiting deployment.

```
<!-- Portfolio position -->
<Position>
  <UniqueID>IE00B4K48X80</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">2000250.00</Amount>
  </TotalValue>
  <TotalPercentage>0.43</TotalPercentage>
  <PriceDate>2026-03-31</PriceDate>
  <ShareClass>
    <Shares>22500</Shares>
    <Price>
      <Amount ccy="EUR">88.90</Amount>
    </Price>
    <PurchaseValue>
      <Amount ccy="EUR">1987420.00</Amount>
    </PurchaseValue>
  </ShareClass>
</Position>

<!-- Asset entry -->
<Asset>
  <UniqueID>IE00B4K48X80</UniqueID>
  <Identifiers>
    <ISIN>IE00B4K48X80</ISIN>
  </Identifiers>
  <Currency>EUR</Currency>
  <Country>IE</Country>
  <Name>iShares Core MSCI Europe UCITS ETF EUR (Acc)</Name>
  <AssetType>SC</AssetType>
  <AssetDetails>
    <ShareClass>
      <Issuer>
        <Identifiers>
          <LEI>549300MS535KC2WH4Z30</LEI>
        </Identifiers>
        <Name>BlackRock Asset Management Ireland Limited</Name>
        <BusinessCountry>IE</BusinessCountry>
        <Type>Non-MMF investment funds</Type>
```

```

    </Issuer>
    <StockMarket>XETR</StockMarket>
    <Listing>A</Listing>
    <ListingUnit>U</ListingUnit>
    <IssueDate>2009-09-29</IssueDate>
    <ExchangeTradedFundFlag>true</ExchangeTradedFundFlag>
  </ShareClass>
</AssetDetails>
<Classifications>
  <Classification>
    <ListedGroup>MORNINGSTAR</ListedGroup>
    <Type>Global Category</Type>
    <Language>en</Language>
    <Value>Europe Large-Cap Blend Equity</Value>
  </Classification>
</Classifications>
</Asset>

```

Three points about this example. First, the `<ShareClass>` child at the position level carries `<Shares>` (the number of fund units held, `xs:decimal`), `<Price>` (the last NAV of the held share class, in its native currency), and `<PurchaseValue>` (the historical cost). Unlike `<Equity>`, the share-class inner block has no `<MarketValue>` field — the consolidated value of the position lives at the `<Position>/<TotalValue>` level, not inside `<ShareClass>`. A producer that wants to carry the held fund's own NAV-times-units figure for debugging purposes can do so in `<OtherPrices>`.

Second, the asset entry marks the held fund as an ETF through `<ExchangeTradedFundFlag>true</ExchangeTradedFundFlag>` inside `<AssetDetails>/<ShareClass>` — a dedicated boolean that is *in addition to* the generic `Listing` enumeration. Consumers that want to identify ETFs read the flag directly rather than parsing the MIC code or the listing status. The `<Classifications>` block carries the Morningstar global category — a real Morningstar label that Europe-large-cap blend ETFs actually occupy — through the idiomatic `<ListedGroup>MORNINGSTAR</ListedGroup>` path.

Third, the holding fund's own cost disclosure under MiFID II does need to aggregate the held ETF's ongoing charges on a proportional basis, but the mechanism is not through a field on the share-class instrument. The aggregation happens in the regulatory templates of Chapter 8, where the EMT provides its own per-instrument cost structure and the consumer rolls them up. For the purposes of the portfolio block, the held ETF is just another position valued at its current NAV.

6.9 Derivatives

Derivatives are the most conceptually complex asset class in FundsXML, and they deserve the pages they are about to receive. For the Europa Growth Fund, derivatives appear in a single place: the two currency forwards that implement the CHF hedge for the R-CHF-ACC-HEDGED share class from Chapter 5. Other funds hold futures, options, and swaps in much larger proportions, and the general framework this section establishes applies to all of them.

6.9.1 Notional versus Market Value — the Key Concept

Derivatives differ from every other asset class because they have *two* value figures, and the figures can differ by orders of magnitude.

Notional is the *economic exposure* of the contract — the amount on which the contract's payoffs are based. A currency forward to buy 59 million CHF and sell 56.7 million EUR at a rate of 1.0420 has a notional of 56.7 million EUR on the sell side and 59 million CHF on the buy side — two valid representations of the same contract, related by the agreed forward rate. A futures contract on 10,000 units of a DAX index future has a notional equal to the index level times the contract multiplier times 10,000. A swap has a notional equal to the principal on which the periodic payments are computed.

Market value (also called *fair value*) is the *present value of the contract itself* as a tradable instrument — the amount a market participant would pay today to take the contract off the fund's hands. When a contract is first entered into, its market value is zero (because both parties agreed it was fair). As the underlying moves, the market value moves away from zero: a currency forward to sell EUR becomes valuable to the seller if the EUR weakens against the bought currency, and worthless or negative if the EUR strengthens. The absolute value of the market value is typically tiny compared to the notional — perhaps one-hundredth to one-thousandth of it, depending on the time remaining to expiry and the volatility of the underlying.

The difference matters enormously. The Europa Growth Fund's CHF hedge forwards have notional amounts in the tens of millions of euros but market values in the hundreds of thousands or less. A consumer that reads the notional and treats it as the position's contribution to market value will overstate the fund's portfolio by tens of millions of euros, and the fund's NAV calculation will be impossible to reconcile.

Where the two numbers live in FundsXML. The schema does not have dedicated *Notional* and *MarketValue* fields at the position level — this is the single most important difference from the generic derivative model a newcomer might expect. Instead:

- The **fair value** of the contract, in the fund's base currency, lives in the position's mandatory `<TotalValue>` element (schema type `FundAmountType`, at the top of `<Position>`, as for every other asset class). For an FX forward, `TotalValue` is the mark-to-market of the contract, and can be positive or negative depending on whether the trade is in or out of the money.
- The **notional amounts** live on the asset side, inside `AssetDetails/<FXForward>/<AmountBuy>` and `<AmountSell>`, paired with the two currencies (`CurrencyBuy` and `CurrencySell`). These are the contractual quantities of each currency, not valuations — they do not change over the life of the contract, while `TotalValue` moves with the spot rate.
- An optional `<Exposures>` container at the position level (schema type `ExposureType`) carries additional exposure figures computed under specific regulatory methodologies — the AIFMD commitment approach, the gross commitment approach, a custom producer method — each as a labelled `<Exposure>` entry with its own amount. This is the field producers populate when they need to carry a notional-style exposure figure explicitly rather than leaving it implicit in the asset-side `AmountSell`.

The discipline is therefore subtly different from the "Notional vs MarketValue" framing one finds in textbook descriptions. A consumer reads `<TotalValue>` for the mark-to-market contribution to the fund's NAV, reads the asset-side `AmountBuy / AmountSell` for the contractual notional amounts, and reads

`<Exposures>/<Exposure>` for any regulatory exposure figure the producer chose to publish. Substituting one for another is still a catastrophic error — it is just that the labels the schema uses are different from the ones the textbook story suggests.

Table 6.5 — Key derivative-related fields, where they live

Purpose	Field location	Type
Mark-to-market in fund currency	Position/TotalValue	FundAmountType
Regulatory exposure	Position/Exposures/Exposure	labelled Type + Value
Asset-class discriminator	Asset/AssetType	2-char: FX / OP / FU / SW
Position-side sub-block	Position/FXForward /Option /Future /Swap	inline type
Contract details	Asset/AssetDetails/FXForward /Option /Future /Swap	FXForward carries ForeignExchangeTradeType
Contract currencies and notionals (FX)	AssetDetails/FXForward/CurrencyBuy + AmountBuy + CurrencySell + AmountSell	decimal + ISO currency
Agreed forward rate (FX)	AssetDetails/FXForward/AgreedFxRate	xs:decimal
Maturity (FX)	AssetDetails/FXForward/MaturityDate	xs:date (mandatory)
Counterparty	AssetDetails/FXForward/Counterparty	CompanyType (mandatory for FX, must have LEI)
Strike (options)	AssetDetails/Option/StrikePrice	xs:decimal
Contract size (futures, options)	AssetDetails/Future/ContractSize / AssetDetails/Option/ContractSize	xs:decimal

6.9.2 FX Forwards

The derivative type the Europa Growth Fund actually holds, and the simplest class of all.

A currency forward is a contract to exchange a fixed amount of one currency for a fixed amount of another currency at a specific date in the future, at a rate agreed when the contract was entered into. In FundsXML, an FX forward is modelled as `AssetType=FX`, and the contract details live in `AssetDetails/<FXForward>`, which is an element of schema type `ForeignExchangeTradeType` (XSD §25529). Its mandatory children, in order, are:

- `<CurrencyBuy>` — the currency the fund is contractually buying (ISO 4217 code). An optional `@isBaseCCY` boolean attribute flags this as the fund's base currency.

- `<AmountBuy>` — the amount in the buy currency, as a plain `xs:decimal`. Note that this is **not** an `<Amount ccy="...">` wrapper; it is a bare decimal, with the currency given by the sibling `<CurrencyBuy>` element. This is a deliberate asymmetry with the position-side amounts, driven by the schema's choice to model the FX trade as a contract rather than a valuation.
- `<CurrencySell>` — the currency the fund is contractually selling.
- `<AmountSell>` — the amount in the sell currency, again as a plain `xs:decimal`.
- `<AgreedFxRate>` — the forward rate agreed at trade time, as a decimal (e.g. `1.0420` for EUR/CHF at 1.0420 Swiss francs per euro).
- `<StartDate>` (optional) — the trade date.
- `<MaturityDate>` — the value date / settlement date. **Mandatory**; this is the date on which the actual currency exchange takes place.
- `<Counterparty>` — a `CompanyType` block naming the bank on the other side of the trade, with its LEI inside `<Identifiers>`. **Mandatory**; every FX forward in FundsXML must name its counterparty.
- `<Deliverable>` (optional, `xs:boolean`) — whether the currencies will be physically exchanged at maturity (*true*) or only the profit/loss will be transferred (*false*, a non-deliverable forward or *NDF*).

The notional of an FX forward is thus **two amounts**, not one: the `AmountBuy` and the `AmountSell`. Producers pick whichever side corresponds to the fund's base currency as the "economic" notional and ignore the other; for a CHF-hedge forward on a EUR-base fund, the EUR-side `AmountSell` (the amount the fund has committed to sell) is the operational notional.

The market value of an FX forward at any time between trade and maturity is a function of the current spot rate and the agreed forward rate: when the current rate is favourable to the fund's side of the trade, the value is positive; when unfavourable, it is negative. The producer computes the mark-to-market using standard present-value techniques and writes the result into the **position's** `<TotalValue>` in the fund's base currency. Consumers do not attempt to recompute.

The position-side `<FXForward>` sub-block carries almost no content. The only fields it defines are `<HedgeRatio>` (an optional `PercentageType` stating how much of the target exposure the forward covers) and `<FxRateForEvaluation>` (an optional `xs:decimal` with the spot rate used for today's valuation). Everything else — the currencies, the amounts, the maturity, the counterparty — lives on the asset side. The reason is that the same contract is usually described once per delivery in `AssetMasterData` and may be held by several funds in the same file; the per-position data is only the fund-specific hedge ratio and today's valuation.

The Europa Growth Fund's CHF hedge consists of one or two overlapping forward contracts at any time: a currently-active forward that was rolled from the previous month and is due to settle at the end of the current month, and sometimes a just-entered forward for the next period. The structure is simple: approximately 56.7 million EUR (equal to the EUR equivalent of the R-CHF-ACC-HEDGED class's total net assets from \$5.12) is sold forward against CHF at a forward rate close to the current spot. The forward is rolled at the end of each month, closing out the old contract and opening a new one.

6.9.3 Futures, Options, and Swaps

The remaining derivative types appear more briefly because they are not held by the Europa Growth Fund. Each gets a short paragraph naming its distinguishing fields.

Futures (`AssetType=FU` , asset-side details in `AssetDetails/<Future>` with schema type `FutureType`) are standardised, exchange-listed derivative contracts. Unlike forwards, they are margined daily — the fund pays or receives a cash amount each day equal to the change in the contract's mark-to-market value, so the position's `<TotalValue>` is conventionally zero or close to it (yesterday's margin movement has already moved into the cash balance). The `FutureType` fields that matter are `<Type>` (an enumeration covering `BF` Bond Future, `CF` Currency Future, `EF` Equity Future, `FRA` Forward Rate Agreement, `IF` Index Future, `IRF` Interest Rate Future, `MMF` Money Market Future, and `OTHER`), `<ContractSize>` , `<MaturityDate>` , and `<BasePrice>` . The position-side `<Future>` sub-block carries `<Contracts>` (the number of contracts, which may be negative for a short position) and a few supporting fields.

Options (`AssetType=OP` , asset-side details in `AssetDetails/<Option>` with schema type `OptionType`) carry additional fields because they are asymmetric contracts. `OptionType/<Type>` distinguishes the option family through a twelve-value enumeration: `BFO` Bond Future Option, `BO` Bond Option, `CFO` Currency Future Option, `EFO` Equity Future Option, `CO` Currency Option, `IFO` Index Future Option, `IO` Index Option, `IRFO` Interest Rate Future Option, `IRO` Interest Rate Option, `OTCIB` OTC Index-basket Option, `SO` Stock Option, `OTHER` . Strike price lives in `OptionType/<StrikePrice>` , contract size in `<ContractSize>` , expiry in `<MaturityDate>` , the call/put distinction in `<CallPutIndicator>` (`C` or `P`), and the exercise style in `<ExecutionStyle>` (American or European). The position-side `<Option>` sub-block carries `<Contracts>` (negative for a written option) and `<Price>` (the option premium). The market value of a long option is its premium times the number of contracts; the notional is the strike times the contract size times the number of contracts. A short option contributes the premium negatively.

Swaps (`AssetType=SW` , asset-side details in `AssetDetails/<Swap>` with schema type `SwapType`) are a family rather than a single type. `SwapType/<Type>` is an enumeration that covers `Currencyswap` , `Crosscurrencyswap` , `Creditdefaultswap` , `Indexswap` , `Totalreturnswap` , `Assetswap` , `Commodityswap` , `Interestrateswap` , `Inflationswap` , and `Other` . The rest of `SwapType` carries the shared metadata — `StartDate` , `MaturityDate` , `SettlementDate` , `Counterparty` , `FXRate` , `AgreedFxRate` — and the per-leg detail lives in a `<Legs>` container holding one or more `<Leg>` entries with their own rate, reference index, and payment-schedule fields. The position-side `<Swap>` sub-block carries `<HedgeRatio>` , `<CurrentSpread>` (relevant for Credit Default Swaps), `<PresentValueOfPayments>` , and a `<LegValues>` container with one entry per leg showing the current valuation of each side as `<Type>BUY</Type>` or `<Type>SELL</Type>` plus `<Value>` . For most European equity UCITS readers this chapter's treatment is enough; specialised bond and macro funds will need the detailed derivative documentation referenced in Appendix E.

6.9.4 FX Forward Example

The active CHF hedge forward of the Europa Growth Fund on 31 March 2026, as it would appear in the portfolio block and the asset master data block.

```
<!-- Portfolio position -->
<Position>
  <UniqueID>EGF-FXF-20260430-001</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">-142318.50</Amount>
```

```

</TotalValue>
<FXForward>
  <HedgeRatio>100.00</HedgeRatio>
  <FxRateForEvaluation>1.0395</FxRateForEvaluation>
</FXForward>
</Position>

<!-- Asset entry -->
<Asset>
  <UniqueID>EGF-FXF-20260430-001</UniqueID>
  <Currency>EUR</Currency>
  <Name>EUR/CHF Forward 30/04/2026 (main hedge roll)</Name>
  <AssetType>FX</AssetType>
  <AssetDetails>
    <FXForward>
      <CurrencyBuy>CHF</CurrencyBuy>
      <AmountBuy>59107354.41</AmountBuy>
      <CurrencySell isBaseCCY="true">EUR</CurrencySell>
      <AmountSell>56724871.39</AmountSell>
      <AgreedFxRate>1.0420</AgreedFxRate>
      <StartDate>2026-02-28</StartDate>
      <MaturityDate>2026-04-30</MaturityDate>
      <Counterparty>
        <Identifiers>
          <LEI>02RNE8IBXP4R0TD8PU41</LEI>
        </Identifiers>
        <Name>Société Générale S.A.</Name>
        <BusinessCountry>FR</BusinessCountry>
        <Type>Deposit-taking corporations except the central bank</Type>
      </Counterparty>
      <Deliverable>true</Deliverable>
    </FXForward>
  </AssetDetails>
</Asset>

```

Reading field by field: the position has no `<Currency>` override (the forward is already expressed in the fund's EUR base) and a mandatory `<TotalValue>` of *negative* 142,318.50 EUR, because the EUR has strengthened slightly against the CHF since the forward was entered into on 28 February, making the agreed sell rate slightly unfavourable to the fund. The negative sign on the `<Amount>` value is entirely schema-valid (the underlying `xs:decimal` type accepts negative values), and it is how the schema represents an out-of-the-money derivative position. Inside `<FXForward>`, the `<HedgeRatio>100.00</HedgeRatio>` declares that the forward fully covers its target exposure, and `<FxRateForEvaluation>1.0395</FxRateForEvaluation>` is the spot rate the producer used for today's mark-to-market.

The asset entry identifies the contract as an FX forward through `<AssetType>FX</AssetType>` and places the contract details inside `<AssetDetails>/<FXForward>`. The buy side is 59,107,354.41 CHF; the sell side is 56,724,871.39 EUR with `@isBaseCCY="true"` flagging it as the fund base currency; and the agreed forward rate is 1.0420 (close to the then-prevailing spot). `<MaturityDate>` is the settlement date on which the currencies will actually be exchanged (30 April 2026, one month out from the valuation point). The `<Counterparty>` is a full `CompanyType` block — the schema refuses anything less — with Société Générale's LEI inside `<Identifiers>`

and the ECB sector classification identifying it as a deposit-taking corporation. The `<Deliverable>true</Deliverable>` flag declares that this is a standard deliverable forward, not a non-deliverable forward with cash settlement.

The `UniqueID` is a producer-generated string (`EGF-FXF-20260430-001`) rather than an ISIN, because over-the-counter forwards have no ISIN. The format is conventional: the fund short code, the instrument type, the maturity date, and a sequence number. Because `xs:ID` demands that the value begin with a letter, the `EGF-` prefix is mandatory; a raw numeric code would not pass schema validation.

The linkage to Chapter 5 is worth making explicit. The R-CHF-ACC-HEDGED share class from §5.10.2 carries `<CurrencyHedgedFlag>true</CurrencyHedgedFlag>`. That flag is the *declaration* that the class is currency-hedged; this forward is the *operational mechanism* by which the hedge is implemented. Without the forward, the CHF class would be exposed to EUR/CHF movements; with the forward, those movements are neutralised (imperfectly — the hedge cost absorbs into NAV, as §5.10.2 warned). The `-142,318.50 EUR` `<TotalValue>` is exactly the kind of small mark-to-market movement that gets absorbed into the CHF class's daily NAV.

6.10 Cash, Real Estate, and Other Asset Classes

The catchment section for everything that is neither equity, bond, fund, nor derivative. Three subsections of unequal importance.

6.10.1 Cash Positions

Cash is operationally the simplest asset class but deserves a careful discussion because FundsXML models it through a choice of three distinct asset-type codes, and picking the right one is the first decision.

Three asset-type codes for cash. The schema distinguishes:

- **AC — Account:** an ordinary bank account the fund maintains with its custodian. Details live in `AssetDetails/<Account>` (schema type `AccountType`, XSD §68). This is the most common cash asset class and covers current accounts, nostro accounts, and cash management accounts.
- **FT — Fixed Time Deposit:** a fixed-term deposit, typically thirty or ninety days, earning a rate above the current-account rate. Details live in `AssetDetails/<FixedTimeDeposit>` with schema type `FixedTimeDepositType`, which adds `StartDate`, `MaturityDate`, `MethodForCalculationOfInterest`, and `InterestRate` fields on top of the counterparty block.
- **CM — Call Money:** a short-term call money or call capital position, possibly used as collateral. Details live in `AssetDetails/<CallMoney>` with schema type `CallCapitalType`, which adds fields for `CollateralFlag` (whether the cash is pledged) and `CollateralDirection` (`PAID` or `RECEIVED`).

The generic *CurrentAccount* / *TimeDeposit* / *CashCollateral* trio one sees in textbook descriptions is therefore expressed through three separate asset-type codes, not through a single `CashType` enumeration on a single asset. A producer picks the code that best fits the account's operational nature and populates the corresponding details block.

The AccountType details block is the one most fund administrators actually use, and its children are minimal: an optional `<IndicatorCreditDebit>` (Credit or Debit) naming the account's default balance direction, optional `<AccountNumber>` and `<IBAN>` for regulatory reporting, optional `<InterestRateDebit>` and `<InterestRateCredit>` for overdraft and credit rates, and a **mandatory** `<Counterparty>` block of type `CompanyType` identifying the bank holding the account.

One position per currency. A fund with positions in EUR, GBP, and CHF has three cash positions, each representing one currency balance, each with its own `<Asset>` entry. There is no mechanism for aggregating multi-currency cash into a single position; the model forces the producer to name each currency explicitly. The position's `<Currency>` element and the asset's top-level `<Currency>` element must agree.

Counterparty and concentration limits. The bank holding the cash matters operationally because concentration limits under UCITS and AIFMD are applied against the counterparty, not the currency; a fund that holds 50 million EUR at a single bank has a 50 million EUR concentration against that bank, regardless of what other banks it also uses. For a custodian-held account the counterparty is the custodian itself; for cash collateral it is whichever party holds the collateral account (often a different bank). The counterparty's LEI is the single most important field on the whole cash asset entry.

Position-side fields. At the position level, a cash entry ends with an `<Account>` child (or `<FixedTimeDeposit>`, `<CallMoney>` for the other types). The `<Account>` inner block is the simplest of all choice children: both its children are optional — an `<MarketValue>` (the current balance, in the account's currency) and an `<Interests>` (accrued interest not yet credited). In practice producers populate `<MarketValue>` because it carries the operational balance, but the schema does not insist. There is no `<Units>`, no `<Shares>`, no `<Contracts>` — cash has no unit count, only a balance — and the authoritative fund-currency value lives at the position's `<TotalValue>` as always.

Example: the Europa Growth Fund's primary EUR cash position at its Luxembourg custodian on 31 March 2026.

```
<!-- Portfolio position -->
<Position>
  <UniqueID>EGF-CASH-EUR-001</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">3247819.45</Amount>
  </TotalValue>
  <TotalPercentage>0.70</TotalPercentage>
  <Account>
    <MarketValue>
      <Amount ccy="EUR" isFundCcy="true">3247819.45</Amount>
    </MarketValue>
  </Account>
</Position>

<!-- Asset entry -->
<Asset>
  <UniqueID>EGF-CASH-EUR-001</UniqueID>
  <Currency>EUR</Currency>
  <Country>LU</Country>
  <Name>EUR Current Account – BGL BNP Paribas</Name>
```

```

<AssetType>AC</AssetType>
<AssetDetails>
  <Account>
    <IndicatorCreditDebit>Credit</IndicatorCreditDebit>
    <InterestRateCredit>0.15</InterestRateCredit>
    <Counterparty>
      <Identifiers>
        <LEI>549300CL2K1QU0ZK7T97</LEI>
      </Identifiers>
      <Name>BGL BNP Paribas S.A.</Name>
      <BusinessCountry>LU</BusinessCountry>
      <Type>Deposit-taking corporations except the central bank</Type>
    </Counterparty>
  </Account>
</AssetDetails>
</Asset>

```

Reading this pair: the position says the fund holds a euro current-account balance of 3.25 million euros at its Luxembourg custodian, representing 0.70% of total net assets. The `<Account>` inner block at the position level carries only the same balance as the position's consolidated `<TotalValue>` — for cash, the two are identical by construction, and the apparent redundancy is a schema artefact. The asset entry uses `<AssetType>AC</AssetType>` to signal *Account*, places the counterparty block inside `<AssetDetails>/<Account>` as the schema requires, and populates the counterparty's LEI (the critical field for concentration-limit reporting) inside `<Identifiers>` following the `CompanyType` convention. An optional `<InterestRateCredit>` of 0.15% indicates the rate the custodian is currently paying on the credit balance.

The Europa Growth Fund also holds small balances in GBP (from sterling dividend receipts on UK-listed holdings) and in CHF (for the CHF-hedge operational account). Both follow the same pattern, each as its own position with its own asset entry, both held at the same BGL BNP Paribas custodian.

6.10.2 Real Estate

The Europa Growth Fund holds no real estate, and this chapter does not attempt a detailed treatment. The structural essentials are worth naming briefly so that a reader who one day meets a real-estate fund delivery recognises the pattern.

Real estate comes in two forms. **Indirect real estate** — shares in listed REITs, units in property funds, private real-estate vehicles — is modelled through the ordinary `EQ` or `SC` asset types; the "real estate" nature shows up in the issuer's business and in classifications, not in a new asset class. **Direct real estate** — the fund owns the physical building — is modelled through `AssetType=RE` with `AssetDetails/<RealEstate>`, which carries property-specific fields for location, valuation method, and occupancy. Open-ended German and Luxembourg property funds are the usual populators of this branch; the schema's `RealEstateType` runs to several pages and its detail is outside the scope of this chapter. Readers interested in real-estate-specific reporting should consult the specialised documentation referenced in Appendix E.

6.10.3 Commodities, Private Equity, and Alternatives

A short catch-all for asset classes that do not appear in the Europa Growth Fund and are not treated in detail in this chapter. The schema has dedicated types for each:

- **Commodities** — `AssetType=C0`, details in `AssetDetails/Commodity`. Physical commodities are effectively excluded from UCITS portfolios by eligibility rules; they appear in specialised AIFs. Commodity futures are derivatives (§6.9), commodity-linked structured notes are bonds (§6.7).
- **Private equity** — `AssetType=PE`, details in `AssetDetails/PrivateEquity`. Holdings in non-listed companies, typically held through a limited partnership structure. Valued by mark-to-model rather than by market quote, updated quarterly or semi-annually. AIF territory.
- **Hedge funds** use `AssetType=SC` (share class, like any other fund) but with additional operational complications around redemption gates, lock-up periods, and side pockets that the position-level schema does not model directly. Producers extend through `CustomAttributes`.
- **Rights, warrants, certificates, crypto** — `AssetType=RI / WA / CE / CR` respectively, each with a dedicated details block in `AssetDetailsType`.

None of these asset classes appears in the Europa Growth Fund's portfolio, and none appears in the complete example of §6.12. They are named for completeness and for the reader who one day encounters an AIF delivery and needs to know that the mechanism exists — the branch in `AssetDetailsType` is already there, waiting for the asset.

6.11 Portfolio Breakdowns by Dimension

We close Part III of the chapter with a topic that sits alongside positions rather than inside them: the pre-computed aggregates that FundsXML delivers in its `BreakDowns` element. The capitalisation is deliberate — the element name is `BreakDowns` with a capital D, the container is a child of `Portfolio` (not of `FundDynamicData` or a separate root), and the structure is surprisingly more flexible than most readers expect.

6.11.1 Why Breakdowns Exist

A portfolio with 150 positions can be sliced along many dimensions — by sector, by country, by currency, by rating, by maturity, by instrument type. A consumer that wants to render a sector chart on a factsheet does not want to re-aggregate 150 position records every time the factsheet regenerates; it wants a ready-made list of "Healthcare 17.50%, Financials 17.20%, Industrials 14.40%, ...". The `BreakDowns` element carries exactly this: pre-computed slices of the portfolio along standard dimensions, each containing a list of buckets with labels and percentages.

Two more substantive reasons justify the pre-computation. First, **classification conventions are producer-specific**. Two consumers that re-aggregate the same 150 positions may apply different sector classifications (GICS vs ICB), different country assignments (country of issue vs country of risk), or different lookthrough rules for held funds. The producer, being the single authoritative source, emits *one* classification, and every consumer using the producer's breakdowns sees the same picture. Consumers that want a different classification can still re-aggregate themselves, but the default is to trust the producer.

Second, **some breakdowns cannot be re-computed from positions alone**. A fund's currency breakdown, if it takes hedging into account, cannot be derived from the position-level currency fields alone — it requires knowledge of which forwards hedge which exposures, and the producer computes the net result after applying the hedges. A consumer recomputing from raw positions would get a pre-hedge exposure, which is not what the factsheet shows.

6.11.2 Structure of a Breakdown

The schema type is `BreakDownsType` (XSD §2390), and it is multi-dimensional by design. A single `<BreakDowns>` element contains one or more `<BreakDown>` children; each `<BreakDown>` carries:

- A `<Dimensions>` container holding one or more `<Dimension>` strings naming the dimensions along which the breakdown slices the portfolio. Common values are `Sector`, `Country`, `Currency`, `Rating`, `Duration`, `AssetType`, `Region`, `Size`. A single `<Dimension>` gives a one-dimensional breakdown; two `<Dimension>` entries (`Currency` and `AssetType`) produce a two-dimensional cross-tabulation.
- An optional `<CalculationMethod>` free-text field naming the basis of the calculation (`Market Value`, `Exposure`, `Brutto`, `Netto`, and so on).
- A mandatory `<Values>` container holding one or more `<Value>` entries. Each `<Value>` is one cell of the breakdown table and carries:
 - A `<DimValues>` container with one `<DimValue>` per dimension, each labelled by a `@dim` attribute identifying which dimension the value belongs to. For a one-dimensional sector breakdown, each `<Value>` has one `<DimValue dim="Sector">Financials</DimValue>` element; for a two-dimensional breakdown, each `<Value>` has two `<DimValue>` elements, one per dimension.
- A mandatory `<Percentage>` (`PercentageType`, in per cent) giving the cell's share of the portfolio, from 0 to 100.
- An optional nested `<BreakDowns>` element, allowing a breakdown cell to carry its own sub-breakdowns — the structure is recursive. A *Financials* sector cell, for example, can carry an inner breakdown into *Banks*, *Insurance*, *Diversified Financials*, each with its own percentage relative to the parent cell's share of the portfolio.

The completeness rule is that the sum of percentages across all `<Value>` entries within a single `<BreakDown>` should equal 100, subject to a rounding tolerance of a few basis points. A breakdown that sums to 99.95% or 100.03% is acceptable; a breakdown that sums to 92% is missing a cell and is a data-quality issue.

A concrete example — a one-dimensional sector breakdown for a generic European equity fund, shortened to four cells:

```
<BreakDowns>
  <BreakDown>
    <Dimensions>
      <Dimension>Sector</Dimension>
    </Dimensions>
    <CalculationMethod>Market Value</CalculationMethod>
    <Values>
      <Value>
        <DimValues>
          <DimValue dim="Sector">Financials</DimValue>
        </DimValues>
      </Value>
    </Values>
  </BreakDown>
</BreakDowns>
```



```

        <Percentage>22.50</Percentage>
    </Value>
    <Value>
        <DimValues>
            <DimValue dim="Sector">Industrials</DimValue>
        </DimValues>
        <Percentage>18.75</Percentage>
    </Value>
    <Value>
        <DimValues>
            <DimValue dim="Sector">Consumer Discretionary</DimValue>
        </DimValues>
        <Percentage>15.20</Percentage>
    </Value>
    <Value>
        <DimValues>
            <DimValue dim="Sector">Healthcare</DimValue>
        </DimValues>
        <Percentage>14.15</Percentage>
    </Value>
    <!-- further cells elided -->
</Values>
</BreakDown>
</BreakDowns>

```

A producer typically emits several `<BreakDown>` entries inside one `<BreakDowns>` container — one for each dimension the fund's consumers need. The Europa Growth Fund's monthly delivery emits breakdowns by sector, country, and currency at a minimum, and its complete `<BreakDowns>` block in §6.12 shows all three as sibling `<BreakDown>` entries.

Two-dimensional breakdowns. The multi-dimensional design shines when a consumer needs a cross-tabulation. A `<BreakDown>` with two `<Dimension>` children (*Country* and *Sector*) has one `<Value>` per country-sector cell, each with two `<DimValue>` elements carrying the two labels. The German Financials cell, the French Industrials cell, the Swiss Healthcare cell, and so on — each appears as a separate `<Value>` entry with its own percentage. This flexibility is largely unused in current production pipelines because most consumers prefer simpler one-dimensional breakdowns, but it is there when the need arises.

6.11.3 The Common Breakdown Types in Practice

Sector breakdown uses GICS by default for European equity funds, ICB for some British contexts, NACE for regulatory reporting. A diversified European equity fund typically shows Financials, Industrials, Consumer Discretionary, Healthcare, Technology, and Consumer Staples as the five or six largest sectors, each in the 10% to 25% range, followed by Utilities, Energy, Materials, Real Estate, and Communication Services in smaller allocations. The precise weights shift with the benchmark and the manager's active bets.

Country breakdown is based on the issuer's country of domicile (for equities) or country of risk (for bonds). A typical European equity fund is weighted heavily toward France and Germany — together 40% to 50% — with Switzerland, the Netherlands, Spain, Italy, and Scandinavia filling the remainder. The Europa Growth Fund's breakdown in §6.12 will show exactly this pattern.

Currency breakdown shows the currencies in which the portfolio is denominated, typically before hedging effects. For a European equity fund the dominant currency is EUR, with CHF (Swiss stocks), GBP (UK stocks), and smaller allocations to DKK, NOK, and SEK from Nordic holdings.

Rating breakdown matters for bond portfolios and is largely irrelevant for pure equity funds. The standard cells are AAA, AA+ through AA-, A+ through A-, BBB+ through BBB-, BB and below, Not Rated. The producer picks a convention for which agency's rating to use (typically the middle of the three main agencies, or the lowest for conservatism).

Duration breakdown likewise matters for bonds. Cells are typically 0–1 year, 1–3 years, 3–5 years, 5–10 years, 10–20 years, 20+ years, or a shorter set for short-duration funds. The dimension name in the schema is `Duration`, and the convention is to put the same bucket labels that the factsheet uses into the `<DimValue dim="Duration">` elements. Equity funds do not emit duration breakdowns.

6.11.4 Producer versus Consumer Computation

A consumer that re-computes breakdowns from the raw positions will often find small differences from the producer's figures. Three sources account for almost all of the differences.

Rounding. Position-level percentages are rounded to two decimal places; summing rounded values gives a total that can differ from the producer's figure (which was computed at full precision before rounding) by a few basis points. This is an artefact of rounding, not a data error.

Classification ambiguity. A company like Siemens is both an industrial conglomerate and a technology provider, and different classification schemes place it differently. The producer's breakdown reflects the producer's chosen classification; a consumer using its own classification library will get different numbers. Neither is wrong; they are answers to slightly different questions.

Lookthrough. When the fund holds another fund (§6.8) and the producer has populated `<PositionsDecomposed>` to apply lookthrough to the held fund's underlying holdings, the producer's sector breakdown is computed from the decomposed positions and reflects the underlying holdings rather than the held fund as a single *Fund* cell. A consumer that reads `<Positions>` directly but reads the producer's breakdown will see a different picture than one that re-aggregates `<Positions>` itself. The difference is the lookthrough effect.

The operational rule is straightforward: **consumers trust the producer's breakdowns and recompute only for validation purposes, with appropriate tolerance.** Chapter 10 will formalise the tolerances that production validators should apply to breakdown reconciliation.

6.12 The Complete Europa Growth Fund Portfolio

We are now ready to assemble the largest example in the book. The Europa Growth Fund on 31 March 2026 holds approximately 150 distinct positions: 145 European equity positions across eleven countries and eleven sectors, three cash positions (EUR, GBP, CHF), and two currency forwards implementing the CHF hedge. The complete portfolio appears in Appendix D; this section shows a representative sample of fifteen equity positions, the three cash positions, and the two forwards, together with their `AssetMasterData` entries and the resulting portfolio breakdowns.

6.12.1 The Real Portfolio

Before looking at the XML, a word on what the real portfolio looks like. The fund's 145 equity positions are not uniform in size; the largest ten or so positions account for thirty to forty percent of the portfolio (the manager's conviction picks), while the remaining 130-plus positions form a long tail of smaller holdings for diversification. The sample of fifteen positions shown below is drawn from the larger end of the portfolio, so that each position is meaningful in size (between five and fifteen million euros), and is distributed across sectors and countries so that the portfolio breakdowns at the end of the section are meaningful.

The combined market value of the fifteen sample positions plus the three cash positions comes to approximately 130 million EUR — roughly 28% of the fund's total net assets of 464.55 million EUR. The remaining 72% lives in the 130 positions that are not shown here. The currency forwards do not add materially to the portfolio market value (their market values are in the low hundreds of thousands, positive or negative, absorbed into the hedged class's NAV).

6.12.2 The Portfolio Block

```
<Portfolio>
  <NavDate>2026-03-31</NavDate>
  <Positions>
    <!-- Equity positions (fifteen representative entries) -->
    <Position>
      <UniqueID>DE0007236101</UniqueID>
      <Currency>EUR</Currency>
      <TotalValue><Amount ccy="EUR" isFundCcy="true">8125000.00</Amount></TotalValue>
      <TotalPercentage>1.75</TotalPercentage>
      <PriceDate>2026-03-31</PriceDate>
      <Equity>
        <Units>50000</Units>
        <Price><Amount ccy="EUR">162.50</Amount></Price>
        <MarketValue><Amount ccy="EUR" isFundCcy="true">8125000.00</Amount>
      </MarketValue>
    </Equity>
  </Position>
  <Position>
    <UniqueID>DE0007164600</UniqueID>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">8538000.00</Amount></TotalValue>
    <TotalPercentage>1.84</TotalPercentage>
    <PriceDate>2026-03-31</PriceDate>
    <Equity>
      <Units>60000</Units>
      <Price><Amount ccy="EUR">142.30</Amount></Price>
      <MarketValue><Amount ccy="EUR" isFundCcy="true">8538000.00</Amount>
    </MarketValue>
  </Equity>
</Position>
<Position>
  <UniqueID>DE0008404005</UniqueID>
```

```

    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">7964800.00</Amount></TotalValue>
    <TotalPercentage>1.72</TotalPercentage>
    <PriceDate>2026-03-31</PriceDate>
    <Equity>
      <Units>32000</Units>
      <Price><Amount ccy="EUR">248.90</Amount></Price>
      <MarketValue><Amount ccy="EUR" isFundCcy="true">7964800.00</Amount>
</MarketValue>
    </Equity>
  </Position>
  <!-- LVMH, TotalEnergies, BNP Paribas (FR) and Nestlé, Roche, Novartis (CH)
    follow the same pattern; the three Swiss positions carry their prices in
    CHF and their TotalValue in EUR (isFundCcy="true") alongside a CHF-native
    Amount for transparency -->
  <Position>
    <UniqueID>CH0038863350</UniqueID>
    <Currency>CHF</Currency>
    <TotalValue>
      <Amount ccy="EUR" isFundCcy="true">6017500.00</Amount>
      <Amount ccy="CHF">6270290.00</Amount>
    </TotalValue>
    <TotalPercentage>1.30</TotalPercentage>
    <PriceDate>2026-03-31</PriceDate>
    <Equity>
      <Units>65000</Units>
      <Price><Amount ccy="CHF">96.40</Amount></Price>
      <MarketValue>
        <Amount ccy="EUR" isFundCcy="true">6017500.00</Amount>
        <Amount ccy="CHF">6270290.00</Amount>
      </MarketValue>
    </Equity>
  </Position>
  <!-- ASML (NL), Shell, Unilever, AstraZeneca (GB), Iberdrola, Banco Santander (ES)
    follow the pattern. Cash positions come next. -->

  <!-- Cash accounts (one position per currency) -->
  <Position>
    <UniqueID>EGF-CASH-EUR-001</UniqueID>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">3247819.45</Amount></TotalValue>
    <TotalPercentage>0.70</TotalPercentage>
    <Account>
      <MarketValue><Amount ccy="EUR" isFundCcy="true">3247819.45</Amount>
</MarketValue>
    </Account>
  </Position>
  <Position>
    <UniqueID>EGF-CASH-GBP-001</UniqueID>
    <Currency>GBP</Currency>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">208541.30</Amount></TotalValue>
    <TotalPercentage>0.04</TotalPercentage>

```

```

    <Account>
      <MarketValue><Amount ccy="EUR" isFundCcy="true">208541.30</Amount>
</MarketValue>
    </Account>
  </Position>
<Position>
  <UniqueID>EGF-CASH-CHF-001</UniqueID>
  <Currency>CHF</Currency>
  <TotalValue><Amount ccy="EUR" isFundCcy="true">156217.80</Amount></TotalValue>
  <TotalPercentage>0.03</TotalPercentage>
  <Account>
    <MarketValue><Amount ccy="EUR" isFundCcy="true">156217.80</Amount>
</MarketValue>
  </Account>
</Position>

<!-- FX forwards implementing the CHF hedge for R-CHF-ACC-HEDGED -->
<Position>
  <UniqueID>EGF-FXF-20260430-001</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue><Amount ccy="EUR" isFundCcy="true">-142318.50</Amount></TotalValue>
  <FXForward>
    <HedgeRatio>100.00</HedgeRatio>
    <FxRateForEvaluation>1.0395</FxRateForEvaluation>
  </FXForward>
</Position>
<Position>
  <UniqueID>EGF-FXF-20260430-002</UniqueID>
  <Currency>EUR</Currency>
  <TotalValue><Amount ccy="EUR" isFundCcy="true">18420.75</Amount></TotalValue>
  <FXForward>
    <HedgeRatio>100.00</HedgeRatio>
    <FxRateForEvaluation>1.0395</FxRateForEvaluation>
  </FXForward>
</Position>
</Positions>
<!-- BreakDowns follow in §6.12.4 -->
</Portfolio>

```

6.12.3 The AssetMasterData Block (Representative Entries)

At the root level of the document, the `AssetMasterData` block carries one entry per unique position referenced above. We show three representative entries in full — a German equity, a Swiss equity, and a cash account — together with one FX forward, and list the remainder in summary form.

```

<AssetMasterData>
  <Asset>
    <UniqueID>DE0007236101</UniqueID>
    <Identifiers>

```

```

    <ISIN>DE0007236101</ISIN>
    <GermanWKN>723610</GermanWKN>
  </Identifiers>
  <Currency>EUR</Currency>
  <Country>DE</Country>
  <Name>Siemens AG</Name>
  <AssetType>EQ</AssetType>
  <AssetDetails>
    <Equity>
      <StockMarket>XETR</StockMarket>
      <Issuer>
        <Identifiers><LEI>W38RGI023J3WT1HWRP32</LEI></Identifiers>
        <Name>Siemens Aktiengesellschaft</Name>
        <BusinessCountry>DE</BusinessCountry>
        <Type>Non-financial corporations</Type>
      </Issuer>
      <Listing>A</Listing>
      <CustomMarketCapitalization>
        <Name>Large Cap</Name>
        <Date>2026-03-31</Date>
        <Value>145000000000</Value>
      </CustomMarketCapitalization>
      <Industries>
        <IndustryCode>
          <Name>GICS</Name>
          <Date>2026-03-31</Date>
          <Value>20106010</Value>
        </IndustryCode>
      </Industries>
    </Equity>
  </AssetDetails>
  <Classifications>
    <Classification>
      <UnlistedGroup>GICS</UnlistedGroup>
      <Type>Sector</Type>
      <Language>en</Language>
      <Value level="1">Industrials</Value>
      <Value level="2">Industrial Conglomerates</Value>
    </Classification>
  </Classifications>
</Asset>

<Asset>
  <UniqueID>CH0038863350</UniqueID>
  <Identifiers>
    <ISIN>CH0038863350</ISIN>
    <SwissValorenCode>3886335</SwissValorenCode>
  </Identifiers>
  <Currency>CHF</Currency>
  <Country>CH</Country>
  <Name>Nestlé S.A.</Name>
  <AssetType>EQ</AssetType>

```

```

<AssetDetails>
  <Equity>
    <StockMarket>XVTX</StockMarket>
    <Issuer>
      <Identifiers><LEI>549300DLLUD0PKVWST52</LEI></Identifiers>
      <Name>Nestlé S.A.</Name>
      <BusinessCountry>CH</BusinessCountry>
    </Issuer>
  </Equity>
</AssetDetails>
<Classifications>
  <Classification>
    <UnlistedGroup>GICS</UnlistedGroup>
    <Type>Sector</Type>
    <Language>en</Language>
    <Value>Consumer Staples</Value>
  </Classification>
</Classifications>
</Asset>

<Asset>
  <UniqueID>EGF-CASH-EUR-001</UniqueID>
  <Currency>EUR</Currency>
  <Country>LU</Country>
  <Name>EUR Current Account – BGL BNP Paribas</Name>
  <AssetType>AC</AssetType>
  <AssetDetails>
    <Account>
      <IndicatorCreditDebit>Credit</IndicatorCreditDebit>
      <InterestRateCredit>0.15</InterestRateCredit>
      <Counterparty>
        <Identifiers><LEI>549300CL2K1QU0ZK7T97</LEI></Identifiers>
        <Name>BGL BNP Paribas S.A.</Name>
        <BusinessCountry>LU</BusinessCountry>
        <Type>Deposit-taking corporations except the central bank</Type>
      </Counterparty>
    </Account>
  </AssetDetails>
</Asset>

<!-- Remaining equity asset entries (summary): -->
<!-- SAP SE (DE0007164600, LEI 529900D6BF99LW9F3487, IT, DE, XETR) -->
<!-- Allianz SE (DE0008404005, LEI 529900K9B0N5BT694847, Insurance, DE, XETR) -->
<!-- LVMH (FR0000121014, LEI 96950065LBWY0APQIM86, Cons. Discr., FR, XPAR) -->
<!-- TotalEnergies (FR0000120271, LEI 529900S21EQ1B04ESM68, Energy, FR, XPAR) -->
<!-- BNP Paribas (FR0000131104, LEI R0MUWSFPU8MPR08K5P83, Financials, FR, XPAR) -->
<!-- Roche (CH0012032048, LEI 549300D30HVV27GBHI46, Healthcare, CH, XVTX) -->
<!-- Novartis (CH0012005267, LEI 549300D9YI2R72XBBM14, Healthcare, CH, XVTX) -->
<!-- ASML (NL0010273215, LEI 724500Y6DUVHQD60XN27, IT, NL, XAMS) -->
<!-- Shell (GB00BP6MXD84, LEI 21380068P1DRHMJ8KU70, Energy, GB, XLON) -->
<!-- Unilever (GB00B10RZP78, LEI 549300MKFYEKVRWML317, Cons. Staples, GB, XLON) -->
<!-- Iberdrola (ES0144580Y14, LEI K2RJR3UR2UQIYOV1YS65, Utilities, ES, XMAD) -->

```

```

<!-- Santander (ES0113900J37, LEI 5493006QMFDDMYWIAM13, Financials, ES, XMAD) -->
<!-- AstraZeneca (GB0009895292, LEI 213800ZCSBQL7LI8DI47, Healthcare, GB, XLON)-->

<!-- Cash asset entries (summary): -->
<!-- EGF-CASH-GBP-001 (AC, GBP, custodian BGL BNP Paribas LEI 549300CL2K1QU0ZK7T97) -
->
<!-- EGF-CASH-CHF-001 (AC, CHF, custodian BGL BNP Paribas LEI 549300CL2K1QU0ZK7T97) -
->

<!-- FX Forward asset entries -->
<Asset>
  <UniqueID>EGF-FXF-20260430-001</UniqueID>
  <Currency>EUR</Currency>
  <Name>EUR/CHF Forward 30/04/2026 (main hedge roll)</Name>
  <AssetType>FX</AssetType>
  <AssetDetails>
    <FXForward>
      <CurrencyBuy>CHF</CurrencyBuy>
      <AmountBuy>59107354.41</AmountBuy>
      <CurrencySell isBaseCCY="true">EUR</CurrencySell>
      <AmountSell>56724871.39</AmountSell>
      <AgreedFxRate>1.0420</AgreedFxRate>
      <StartDate>2026-02-28</StartDate>
      <MaturityDate>2026-04-30</MaturityDate>
      <Counterparty>
        <Identifiers><LEI>02RNE8IBXP4R0TD8PU41</LEI></Identifiers>
        <Name>Société Générale S.A.</Name>
        <BusinessCountry>FR</BusinessCountry>
        <Type>Deposit-taking corporations except the central bank</Type>
      </Counterparty>
      <Deliverable>true</Deliverable>
    </FXForward>
  </AssetDetails>
</Asset>
<!-- Second FX forward (EGF-FXF-20260430-002) is structurally identical with
      AmountBuy=2918000.00, AmountSell=2800000.00, AgreedFxRate=1.0421,
      StartDate=2026-03-15, same MaturityDate and counterparty. -->
</AssetMasterData>

```

The elided entries follow the same pattern as the fully-shown ones; each equity entry carries its `<Identifiers>` block with the ISIN (and where present the GermanWKN or SwissValorenCode), its `<Currency>`, its `<Country>`, its `<Name>`, its `AssetType=EQ`, and an `<AssetDetails>/<Equity>` block with the stock market MIC, the issuer's LEI inside a `CompanyType` block, the listing status, and an optional `<CustomMarketCapitalization>` size bucket; each cash entry carries its counterparty in `<AssetDetails>/<Account>`; the second FX forward is structurally identical to the first with different amounts and rates. Appendix D prints the full block with every entry populated.

6.12.4 Portfolio Breakdowns for the Sample

Computed from the fifteen equity positions shown above (scaled to 100% of the equity portion, excluding cash and forwards), three breakdowns are representative. They sit inside the same `<Portfolio>` element as the positions, as a sibling of `<Positions>` rather than at the root.

```
<BreakDowns>
  <BreakDown>
    <Dimensions><Dimension>Sector</Dimension></Dimensions>
    <CalculationMethod>Market Value</CalculationMethod>
    <Values>
      <Value>
        <DimValues><DimValue dim="Sector">Healthcare</DimValue></DimValues>
        <Percentage>17.50</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Financials</DimValue></DimValues>
        <Percentage>17.20</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Industrials</DimValue></DimValues>
        <Percentage>14.40</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Technology</DimValue></DimValues>
        <Percentage>14.30</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Consumer Staples</DimValue></DimValues>
        <Percentage>11.90</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Consumer Discretionary</DimValue></DimValues>
        <Percentage>9.80</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Energy</DimValue></DimValues>
        <Percentage>8.90</Percentage>
      </Value>
      <Value>
        <DimValues><DimValue dim="Sector">Utilities</DimValue></DimValues>
        <Percentage>6.00</Percentage>
      </Value>
    </Values>
  </BreakDown>
  <BreakDown>
    <Dimensions><Dimension>Country</Dimension></Dimensions>
    <CalculationMethod>Market Value</CalculationMethod>
    <Values>
      <Value>
        <DimValues><DimValue dim="Country">GB</DimValue></DimValues>
```



```

    <Percentage>24.70</Percentage>
  </Value>
  <Value>
    <DimValues><DimValue dim="Country">FR</DimValue></DimValues>
    <Percentage>20.20</Percentage>
  </Value>
  <Value>
    <DimValues><DimValue dim="Country">DE</DimValue></DimValues>
    <Percentage>19.90</Percentage>
  </Value>
  <Value>
    <DimValues><DimValue dim="Country">CH</DimValue></DimValues>
    <Percentage>13.40</Percentage>
  </Value>
  <Value>
    <DimValues><DimValue dim="Country">ES</DimValue></DimValues>
    <Percentage>12.00</Percentage>
  </Value>
  <Value>
    <DimValues><DimValue dim="Country">NL</DimValue></DimValues>
    <Percentage>9.80</Percentage>
  </Value>
</Values>
</BreakDown>
<BreakDown>
  <Dimensions><Dimension>Currency</Dimension></Dimensions>
  <CalculationMethod>Market Value</CalculationMethod>
  <Values>
    <Value>
      <DimValues><DimValue dim="Currency">EUR</DimValue></DimValues>
      <Percentage>61.90</Percentage>
    </Value>
    <Value>
      <DimValues><DimValue dim="Currency">GBP</DimValue></DimValues>
      <Percentage>24.70</Percentage>
    </Value>
    <Value>
      <DimValues><DimValue dim="Currency">CHF</DimValue></DimValues>
      <Percentage>13.40</Percentage>
    </Value>
  </Values>
</BreakDown>
</BreakDowns>

```

The breakdowns above are computed from the fifteen-position sample. The breakdowns for the full 150-position portfolio in Appendix D are somewhat different: the UK weighting, in particular, is far less pronounced in the real portfolio (our sample overweights UK blue chips for illustrative purposes), and the sector distribution is more evenly spread across Financials, Industrials, Consumer, and Healthcare without the healthcare dominance that our sample shows. Note also that the country breakdown uses ISO 3166-1 alpha-2 codes (DE , FR , CH , GB , ES , NL) as `DimValue` labels — a producer convention that keeps the breakdown language-independent; a factsheet engine that

wants to render "Germany" rather than "DE" applies its own localised mapping at display time. The *structural* lesson is the same either way: a breakdown is a cell-by-cell summary of the portfolio along one or more dimensions, and consumers read it directly rather than re-computing from positions.

6.12.5 A Three-Pass Reading

The example is best understood in three passes, each focusing on one of the asset classes the fund holds.

First pass — equities. Each of the fifteen equity positions follows the pattern of §6.6: a `<Position>` with mandatory `<UniqueID>` (the ISIN), optional `<Currency>` (the instrument's native currency), mandatory `<TotalValue>` (the consolidated value in the fund's EUR base, flagged `isFundCcy="true"`), optional `<TotalPercentage>` (the position's weight), optional `<PriceDate>`, and a mandatory `<Equity>` child carrying the per-unit detail fields (`<Units>`, `<Price>`, `<MarketValue>`). The asset entries populate the instrument's identifiers, the `<Equity>` details block with issuer LEI and stock market, and one or more `<Classifications>` entries. Three of the fifteen positions are Swiss companies priced in CHF (Nestlé, Roche, Novartis) and three are British companies priced in GBP (Shell, Unilever, AstraZeneca); the remaining nine are priced in EUR. The `<TotalValue>` on every position is EUR-tagged with `isFundCcy="true"`, with the non-EUR positions carrying a second `<Amount>` in their native currency for transparency.

Second pass — cash. Three cash positions, one per currency, each with `AssetType=AC` in the master-data entry and an `<Account>` child at both the position level and inside `AssetDetails`. The `<Account>` inner block at the position level contains only a `<MarketValue>` (the account balance), and the consolidated `<TotalValue>` at the position's top level carries the same balance expressed in the fund's base currency. Each cash asset entry names the custodian (BGL BNP Paribas S.A., the Europa Growth Fund's Luxembourg custodian) inside `AssetDetails/Account/Counterparty`, with the counterparty's LEI embedded in `CompanyType/Identifiers`. The GBP and CHF balances are small — residues from dividend receipts and hedge-operational needs — while the EUR balance of 3.24 million is the fund's main working cash.

Third pass — derivatives. Two currency forwards implementing the CHF hedge for the R-CHF-ACC-HEDGED share class. Each appears in `<Positions>` with an almost-empty `<FXForward>` sub-block (just a `<HedgeRatio>` and the spot rate used for today's valuation) and a position-level `<TotalValue>` that is the mark-to-market of the contract in EUR — *negative* 142,318.50 EUR for the main forward and a small positive 18,420.75 EUR for the secondary forward. The actual contract details — who is buying what from whom, at what rate, for settlement on which date, with whose LEI — live on the asset side under `AssetDetails/FXForward`. The main forward has an `AmountSell` of 56.7 million EUR and a matching `AmountBuy` of 59.1 million CHF at the `AgreedFxRate` of 1.0420 (matching the EUR-equivalent of the hedged class's total net assets from Chapter 5). The secondary forward is a much smaller residual covering a change in the class's AUM since the last monthly roll, with `AmountSell` 2.8 million EUR. Together the two forwards represent the hedge that makes the `<CurrencyHedgedFlag>true</CurrencyHedgedFlag>` on the share class operational, and their `TotalValue` figures absorb directly into the CHF class's NAV as part of its daily valuation.

The sum of `<TotalValue>` across the fifteen equities, three cash positions, and two forwards is approximately 116.8 million EUR — roughly 25% of the fund's total net assets of 464.55 million EUR. The remaining 75% lives in the 130 equity positions that our sample does not show, and the complete portfolio in Appendix D reconciles exactly to the 464.55 million EUR figure. The structural lesson remains the same: every position, every asset entry, every breakdown follows the same patterns established in the earlier sections of this chapter, and those patterns scale from fifteen positions to one hundred and fifty without modification.

6.13 Common Pitfalls

The following short list captures the mistakes that, in our experience, cause the greatest share of portfolio-related production incidents.

- **Schema validation fails at load time because a position references a `UniqueID` that no asset carries.** The parser raises an `xs:IDREF` error — a schema-validation error, not a business-validation error — and the document does not load at all. This is usually the residue of a pipeline in which the position writer and the asset writer ran out of sync. The fix is to emit positions and assets from the same producer pass and to validate every file against the schema before shipping it; Chapter 10 lists this as the first of the mandatory checks.
- **Orphaned `<Asset>` entries in `AssetMasterData`.** Asset entries appear with no position pointing at them — typically the residue of a position that was removed from the portfolio without a cleanup of the master data. Harmless for consumers (the parser does not complain), but a sign of a sloppy producer; pipelines should clean up orphans before emitting.
- **`TotalValue` is missing on a position.** A producer writes the position with `<Equity>/<MarketValue>` but forgets the position-level `<TotalValue>`. The file fails schema validation — `<TotalValue>` is mandatory — and the fund's NAV reconciliation cannot proceed. The fix is to populate `<TotalValue>` on every position without exception, with the position's consolidated market value in the fund's base currency.
- **Bond nominal written as pieces instead of nominal amount.** A producer writes `<Nominal>1000</Nominal>` meaning "one thousand pieces" for a bond whose face value is one million EUR total, when the correct representation is `<Nominal>1000000</Nominal>`. The resulting `Bond/MarketValue` (computed as $\text{Nominal} \times \text{Price} / 100$) is off by a factor of a thousand. The fix is to treat bond `<Nominal>` as a nominal amount without exception and to verify, on every bond position, that $\text{Bond/MarketValue} + \text{Bond/InterestClaimGross} \approx \text{Position/TotalValue}$ to within a rounding tolerance.
- **Market value amount in the wrong currency.** The Nestlé position's price is in CHF, and a careless producer writes `<Equity>/<MarketValue>` in CHF while leaving the position-level `<TotalValue>` empty. A consumer that reads `<TotalValue>` to compute the fund total gets a missing value; a consumer that reads `<Equity>/<MarketValue>` and adds it to the EUR totals of other positions gets an answer that is wrong by the EUR/CHF rate. The fix is to always populate `<TotalValue>` with at least one `<Amount>` in the fund's base currency, tagged `isFundCcy="true"`, and to let the asset-class-specific `<MarketValue>` carry whichever currency suits the producer's internal accounting.
- **FX forward `<TotalValue>` confused with the notional.** A consumer reads the `<AmountSell>` of 56.7 million EUR from `AssetDetails/FXForward` and treats it as the forward's contribution to the fund's total net assets. The fund's NAV suddenly appears to be 56 million EUR larger than reality, and reconciliation fails. The fix is to always read `<Position>/<TotalValue>` for the mark-to-market contribution (which is the small positive or negative mark-to-market value, not the notional) and to use `<AmountBuy>` / `<AmountSell>` only for exposure reporting, not for valuation.
- **`<AssetType>` is written as `Equity` or `Bond` instead of the two-character code.** The schema enforces a two-character enumeration (`EQ` , `B0` , `SC` , `FX` , `AC` , ...) on `<AssetType>` , and anything else is a schema-validation failure at load time. Producers migrating from formats with English type names must run a mapping pass before emitting the file.

- **Breakdown cell percentages do not sum to 100.** A sector breakdown sums to 93% because a cell for *Utilities* is missing from the `<Values>` list even though some positions belong there. The consumer sees the breakdown as incomplete and either refuses to render it or renders it with a suspicious gap. The fix is to emit every non-empty cell and to include an "Other" catch-all `<Value>` for positions that do not fit the standard categories.
- **Cash position without a counterparty LEI.** A cash asset's `AssetDetails/Account/Counterparty` is a `CompanyType`, which requires `<Identifiers>` before `<Name>`. A producer that writes `<Counterparty>` `<Name>BGL BNP Paribas S.A.</Name></Counterparty>` without the `<Identifiers>/<LEI>` child fails schema validation immediately — `Identifiers` is mandatory on `CompanyType`. Even if validation somehow passed, a regulatory consumer could not apply counterparty concentration limits to the position because it cannot identify the counterparty unambiguously. The fix is to populate the counterparty LEI on every cash, derivative, and collateral position without exception.

6.14 Key Takeaways

- Portfolio positions and `AssetMasterData` entries together describe what a fund holds. Positions answer the question *how much*; asset entries answer *what*. Neither is complete without the other, and consumers always read the two structures together.
- The linkage between positions and asset entries runs through the `UniqueID` field, declared `xs:ID` on the asset side and `xs:IDREF` on the position side. The pairing is enforced by the parser at load time, not by business validation at a later stage. The value must conform to XML Name syntax (must not begin with a digit), and the recommended convention is *ISIN when available, deterministic producer-generated string otherwise*, so that the same asset receives the same identifier across consecutive deliveries.
- A `<Position>` has a fixed head of common children — `<UniqueID>`, optional `<Identifiers>`, optional `<Currency>`, mandatory `<TotalValue>`, optional `<TotalPercentage>`, optional cost and FX fields, optional `<PriceDate>`, optional `<PricingSource>` — followed by a **mandatory choice** of exactly one asset-class-specific child from twenty-one options (`<Equity>`, `<Bond>`, `<ShareClass>`, `<Warrant>`, `<Certificate>`, `<Option>`, `<Future>`, `<FXForward>`, `<Swap>`, `<Repo>`, `<FixedTimeDeposit>`, `<CallMoney>`, `<Account>`, `<Fee>`, `<RealEstate>`, `<REIT>`, `<Loan>`, `<Right>`, `<Commodity>`, `<PrivateEquity>`, `<CommercialPaper>`), and optionally a trailing block (`<CapitalYieldsTaxClaim>`, `<InflationaryAdjustment>`, `<RiskCodes>`, `<Underlyings>`, `<CustomAttributes>`). The unit count, per-unit price, and market value of the position live inside the chosen asset-class child (for example `<Equity>/<Units>`, `<Bond>/<Nominal>`, `<ShareClass>/<Shares>`, `<Option>/<Contracts>`), not on the position element itself. There is **no** element called `<Quantity>`, and no element called `<PositionType>` — the complex type `PositionType` is the XSD name of the type of `<Position>`, never a tag in the document.
- An `<Asset>` in `<AssetMasterData>` has its own fixed sequence — `UniqueID`, optional `Identifiers`, optional `DataSupplier`, mandatory `Currency`, optional `IncomeCurrency`, optional `Country`, mandatory `Name`, optional `FISN`, mandatory `AssetType` (as a two-character code), and optional `AssetDetails` carrying the asset-class-specific choice. `Ratings`, `Classifications`, and `CustomAttributes` follow at the bottom.

- For bonds, `<Bond>/<Nominal>` is the nominal value in the issue currency, not the number of pieces. The clean price is expressed as a percentage of nominal, so $\text{MarketValue} \approx \text{Nominal} \times (\text{Price} / 100)$, plus `InterestClaimGross` for the accrued coupon, sums to `Position/TotalValue` in the fund's base currency. There is no `BondType` enumeration; the sovereign-versus-corporate distinction lives on `Issuer/Type` (ECB sector codes) or in `Classifications`.
- For FX forwards, the mark-to-market lives in `Position/TotalValue` and may be positive or negative. The notional amounts live on the asset side in `AssetDetails/FXForward/AmountBuy` and `AmountSell`, paired with `CurrencyBuy` and `CurrencySell`. The position-side `<FXForward>` sub-block is almost empty — just `HedgeRatio` and `FxRateForEvaluation`. The mandatory `Counterparty` on the asset side uses `CompanyType` and must carry an LEI.
- For cash, one position per currency, `AssetType=AC` (or `FT` for fixed-term deposits, `CM` for call money), and the counterparty's LEI on `AssetDetails/Account/Counterparty/Identifiers/LEI`. There is no `CashType` enumeration — the account type is expressed through the two-character `AssetType` code.
- Portfolio breakdowns live in `Portfolio/BreakDowns` (not a separate root-level element) and are multi-dimensional by design. Each `<BreakDown>` names one or more `<Dimension>` strings and carries `<Value>` cells, each with a `<DimValues>` container (one `<DimValue dim="...">` per dimension) and a mandatory `<Percentage>`. The structure is recursive: a cell can carry its own nested `<BreakDowns>`. Trust the producer's breakdowns and tolerate small rounding discrepancies; Chapter 10 formalises the tolerance.
- The Europa Growth Fund's portfolio consists of roughly 150 European equity positions, three cash accounts in three currencies, and two currency forwards implementing the CHF hedge from Chapter 5. The complete representation appears in Appendix D and anchors every fragment treated in the remaining chapters of Part II.

We know what the Europa Growth Fund holds on 31 March 2026. What we do not yet know is *how it got there*. Chapter 7 turns to the events that flow in and out of the fund every business day — subscriptions, redemptions, distributions, and corporate actions — and shows how each of them reshapes the portfolio from one valuation point to the next.



Transactions and Orders

Flows, distributions, and income events

7.1 Setting the Scene: How the Fund Got Where It Is

At the end of Chapter 6 we knew exactly what the Europa Growth Fund held on 31 March 2026 — the one hundred and fifty European equity positions, the three cash accounts, the two currency forwards of the CHF hedge overlay, the 464.5 million euros of total net assets. What we did not know was *how it got there*. Between the previous valuation point on 28 February 2026 and the present one, the fund lived through twenty-three trading days of small and not-so-small changes. Investors bought units and sold them back. A handful of investors moved from one share class to another. European companies paid dividends on the shares the fund holds. Each of these events is a *transaction*, and transactions are the part of fund data that make the dynamic half dynamic.

The chapter walks through the three real schema containers that FundsXML uses to carry these events. The first surprise for a newcomer — and the single most important fact of the chapter — is that **transactions do not live in one central place**. The real schema distributes them across three distinct containers at two different levels of the fund element:

- **ShareClass/Flows/Flow** (schema type `FlowType`) carries subscriptions and redemptions. It lives on the **share-class** side of `SingleFund/ShareClasses`, not inside `FundDynamicData`, because subscriptions and redemptions are by definition share-class-specific events.
- **ShareClass/Distributions/Distribution** (schema type `DistributionType`) carries dividend payout events in which a distributing share class pays its accumulated income out to its investors. Also share-class-specific.
- **Portfolio/Earnings/Earning** (schema type `EarningType`) carries the income events the fund receives from its portfolio holdings — dividends on shares the fund owns, coupons on bonds, interest on cash. These belong to the fund-level **portfolio** inside `FundDynamicData/Portfolios/Portfolio`, because they are properties of the fund's underlying asset pool, not of a specific share class.

Per-security portfolio trading — the portfolio manager buying and selling individual securities — has its own container, `Portfolio/Transactions/Transaction` (schema type `TransactionType`), and is not treated in this chapter; its explicit representation is a topic for Chapter 12 on system landscapes and Chapter 13 on implementation projects. The effect of portfolio trading is visible implicitly in the difference between consecutive portfolio snapshots, which we covered in Chapter 6.

The chapter structure reflects the schema. §7.2 walks through the three containers and their position in the fund element. §7.3 treats the common fields of `FlowType` — the schema type used for subscriptions, redemptions, and the two legs of a switch. §7.4 and §7.5 cover subscriptions and redemptions as the two values of `FlowType/TransactionType` (`SUB` and `RED`). §7.6 treats switches, which the schema represents as a **pair** of linked `Flow` entries rather than as a first-class transaction type. §7.7 treats distributions through

`DistributionType` . §7.8 treats income events through `EarningType` . §7.9 honestly describes the order-execution concepts — cut-off times, forward pricing, swing pricing — that the core schema does **not** model and points at the producer-side mechanisms (`CustomAttributes` , EMT regulatory templates from Chapter 8) for carrying them where they matter. §7.10 assembles the complete transaction block for the Europa Growth Fund's March 2026 period; §7.11 and §7.12 close with pitfalls and takeaways.

One honest disclosure deserves to be made up front. The Europa Growth Fund's I-EUR-DIST share class pays its annual distribution in mid-May, not in March, and the 31 March 2026 delivery therefore contains *no* distribution-paid events. Rather than fabricate one for pedagogical convenience, the chapter treats the absence as a teachable moment: §7.7 shows a hypothetical May distribution to illustrate the structure, and §7.10's complete block honestly omits distributions. This respects both the integrity of the running example and the didactic need to show every event type.

By the end of this chapter, you should be able to:

- identify which of the three schema containers a given transaction belongs to and explain why;
- populate the fields of a `FlowType` entry (subscription or redemption), a `DistributionType` entry, and an `EarningType` entry correctly for a realistic scenario;
- distinguish the four date fields of `FlowType` — `TradeDate` , `AccountingDate` , `SettlementDate` , `ValueDate` — and choose the right one for each operational purpose;
- recognise the difference between distributions *paid* to investors (on `ShareClass/Distributions`) and income *received* from portfolio holdings (in `Portfolio/Earnings`), and never conflate the two;
- represent an intra-fund switch as the two linked `<Flow>` entries the schema actually demands;
- recognise that cut-off times, forward pricing, swing pricing, entry loads, and order-execution variants are operational concepts that the core FundsXML 4.2.8 schema does not model directly, and know where to carry them instead;
- read a complete transaction block for a European UCITS and cross-reference every event to its effect on NAV, shares outstanding, and the portfolio.

7.2 Where Transactions Live in the Schema

Before we look at any transaction in detail, the structural question: *where, inside a FundsXML document, do transactions live?* The answer is surprising to a newcomer and is the single most important thing to internalise before reading the per-type sections.

Transactions are not centralised. There is no single fund-level container for investor flows, distributions, and income events. Each event type lives in its own dedicated container, and each container is attached to a different parent element. The three containers are:

```
<Fund>
  <Identifiers>...</Identifiers>
  <Names>...</Names>
  <Currency>EUR</Currency>
```

```

<SingleFundFlag>true</SingleFundFlag>
<FundDynamicData>
  <Portfolios>
    <Portfolio>
      <NavDate>2026-03-31</NavDate>
      <!-- (1) Income received from portfolio holdings -->
      <Earnings from="2026-03-01" to="2026-03-31">
        <Earning>...</Earning>
        ...
      </Earnings>
    </Portfolio>
  </Portfolios>
</FundDynamicData>
<SingleFund>
  <ShareClasses>
    <ShareClass>
      <Identifiers>...</Identifiers>
      <Names>...</Names>
      <Currency>EUR</Currency>
      <!-- (2) Subscriptions and redemptions on this share class -->
      <Flows from="2026-03-01" to="2026-03-31">
        <Flow>...</Flow>
        ...
      </Flows>
      <!-- (3) Distributions paid by this share class to its investors -->
      <Distributions>
        <Distribution>...</Distribution>
      </Distributions>
    </ShareClass>
    <!-- more share classes, each with their own Flows and Distributions -->
  </ShareClasses>
</SingleFund>
</Fund>

```

Three structural points matter for the rest of the chapter.

First, investor flows and distributions are share-class-specific. A subscription into the R-EUR-ACC class of the Europa Growth Fund goes into the `<Flows>` container of that specific share class, not into a fund-level flow list. A fund with three share classes has three `<Flows>` blocks, each carrying the flows of its own class. Likewise for `<Distributions>`: the distributing I-EUR-DIST class has its own `<Distributions>` block that the two accumulating classes do not share.

Second, portfolio income belongs to the fund-level portfolio, not to any share class. When the fund receives a dividend from an Allianz or BNP Paribas position in its portfolio, the event goes into the fund-level `Portfolio/Earnings` container. All three share classes benefit pro-rata from the income through their share of the fund's total net assets — the income is not re-attributed at the share-class level in the transaction data, only in the NAV roll-forward on valuation day.

Third, the **Flows** container carries two optional date attributes, **@from** and **@to**. They declare the reporting period covered by the aggregated flow data. For a monthly delivery with **ContentDate** 31 March 2026, a typical declaration is `<Flows from="2026-03-01" to="2026-03-31">`. The `<Earnings>` container carries the same pair of attributes for the same purpose. `<Distributions>` does not — a distribution is a single event with its own ex-date, not a period-aggregated collection.

Figure 7.1 — The transaction timeline



Between the two valuation points lie the events that explain the transition: **TotalAssetValues** and **Portfolios** (Chapters 5 and 6) give the *states* at either end, and the three transaction containers collectively give the *transitions* that connect them. A consumer that reconciles the fund's movement from one month to the next reads the opening state from the previous delivery, walks every `<Flow>` in every share class's `<Flows>`, every `<Distribution>` in every share class's `<Distributions>`, and every `<Earning>` in the fund-level `<Earnings>`, and checks that the resulting computed state matches the current delivery's **TotalAssetValues** and **Portfolios**. The reconciliation will almost always have small rounding artefacts, but any material discrepancy is a data-quality signal that deserves investigation.

One short remark to close the section: any of the three containers may legitimately be *empty* or absent — if no subscriptions or redemptions took place on a given class during the reporting period, its `<Flows>` block is either omitted entirely or present with no `<Flow>` children. Consumers should accept empty blocks without treating them as errors. A distribution-less month on the I-EUR-DIST class means no `<Distributions>` block at all — or, equivalently, a `<Distributions>` block with no children. Both are valid.

7.3 The Fields of FlowType

Symmetrical to §6.4 and §6.5 from Chapter 6: the fields that appear on every <Flow> element. Subscriptions, redemptions, and both legs of a switch all use the same schema type, FlowType , and differ only in the value of one mandatory discriminator child. This section walks through the common field set; §§7.4 through 7.6 look at the per-case idioms.

Table 7.1 — Children of <Flow> (schema type FlowType , XSD §25393), in sequence order

Child element	Schema type	Mandatory	Purpose
<ActionCode>	enum C / D / M	yes	Create, Delete, or Modify; C for all new transactions
<DataGroupedBy>	enum (repeatable)	no	Declares whether the data is aggregated by AccountingDate, TradeDate, Channel, InvestorType, etc. Nothing means single-transaction granularity.
<DataGroupedByFrequency>	FrequencyType	no	Period of the aggregation when <DataGroupedBy> names a date field
<TransactionID>	Text256Type	no	Unique identifier within the delivery (stable across deliveries by convention)
<TradeDate>	xs:date	no	Day the order was placed by the investor
<AccountingDate>	xs:date	yes	Day the transaction was booked into the producer's accounting system
<SettlementDate>	xs:date	no	Day cash and units actually change hands
<ValueDate>	xs:date	no	Day the transaction is effective on the customer account
<TransactionType>	enum SUB / RED	yes	The critical discriminator: subscription or redemption — there are only two values
<TransactionSubtype>	xs:string	no	Free-text annotation for sub-variants; used for <i>switch</i> , <i>reinvested dividend</i> , <i>restructuring plan</i> , etc.
<Netted>	xs:boolean	no	Whether the data is already netted against other flows
<Cancellation>	xs:boolean	no	Indicator for cancellation of a previously reported transaction
<OriginalTransactionID>	Text256Type	no	ID of a cancelled transaction, for reversal records
<Units>	xs:decimal	no	Number of units moved. Schema convention: always positive. Direction is expressed through <TransactionType>, not through the sign of <Units>.
<FXRate>	FXRatesType	no	FX rate used when the transaction currency differs from the class currency

Child element	Schema type	Mandatory	Purpose
<TotalValue>	FundAmountType	yes	Total value of the transaction, in the fund's base currency (and optionally also in the class currency)
<Channel>	Text256Type	no	Free text: <i>Retail, Institutional, Wholesale, Web, Distribution partner, ...</i>
<InvestorID>	Text256Type	no	Opaque ID of the investor (hash or internal customer number, never a plain name)
<InvestorTypes>	InvestorTypesType	no	Structured investor-type classification (ECB, EMIR, AIFMD categories)
<InvestorCountry>	ISOCountryCodeType	no	Country of the investor

Four points about the table deserve to be made explicitly in running text rather than left implicit.

Four date fields, not three, and their semantic is subtly different from the textbook model. `FlowType` has `<TradeDate>`, `<AccountingDate>`, `<SettlementDate>`, and `<ValueDate>` — and only `<AccountingDate>` is mandatory. There is no element called `EffectiveDate`. The naming map to the operational concepts goes roughly as follows:

- **<TradeDate>** — the day the investor placed the order, as observed by the distributor. A subscription that reaches the distributor on Monday 16 March 2026 at 11:30 CET has `TradeDate=2026-03-16`.
- **<AccountingDate>** — the day the transaction was booked into the producer's accounting system. In the European forward-pricing model this is typically the same day as the trade date *if* the order arrived before the fund cut-off; orders that arrived after the cut-off get the next trading day's accounting date, and this is how the schema captures the "which NAV did the investor actually pay?" question. `AccountingDate` is the field that maps to the day whose NAV was applied, and it is the one mandatory date on every flow.
- **<SettlementDate>** — the day cash and units actually change hands at T+2 or T+3. A subscription booked on Monday 16 March 2026 settles on Wednesday 18 March (T+2, the Luxembourg standard) or Thursday 19 March (T+3).
- **<ValueDate>** — the day on which the transaction is effective on the investor's account for interest-accrual purposes. Often identical to `AccountingDate` in the European UCITS world, distinct only in specialised structures.

Consumers that compute NAV adjustments read `AccountingDate`; consumers that post cash movements to custodian accounts read `SettlementDate`; consumers that archive audit logs and timestamp orders read `TradeDate`. Only `AccountingDate` is mandatory, but producers should populate all four where they are known, because a consumer that finds one of them missing has to guess. The legacy textbook term *effective date* corresponds to `AccountingDate` in FundsXML vocabulary, and the safest reading discipline is to remember that the schema uses the word *accounting*, not *effective*, for the pricing-date field.

The `<TransactionType>` enumeration has only two values. SUB for subscription, RED for redemption. That is the entire schema-level taxonomy of flow events. There is no dedicated value for *switch*, *distribution*, *income*, or anything else — those event types either live in other containers (`<Distributions>` , `<Earnings>`) or are modelled as a pair of SUB / RED records tied together through `<TransactionSubtype>` and `<TransactionID>` conventions (§7.6 on switches). A producer that invents a `FlowType` enumeration with English labels is writing a different format, not FundsXML.

`<Units>` is always positive on `FlowType` — direction comes from `<TransactionType>` . The schema documentation states this explicitly: "Number of shares/units bought or sold (should be always positive)." A redemption of 15,000 units carries `<Units>15000.0000</Units>` , not `-15000` . The *direction* of the flow — is the fund receiving or paying out cash? — is carried exclusively by `<TransactionType>` . A consumer that infers direction from the sign of `<Units>` is reading a convention that the schema does not define and that many producers will not supply.

`<TotalValue>` is the one mandatory money amount, and it follows the fund-currency convention of Chapter 6. `TotalValue` is of type `FundAmountType` , so it can carry several `<Amount ccy="...">` children at once — one in the fund's base currency (marked `isFundCcy="true"`) and optionally one in the share class's native currency (marked `isShareClassCcy="true"`). For a subscription into the R-CHF-ACC-HEDGED class, a producer who wants to transport both figures writes two `<Amount>` children — one `ccy="CHF" isShareClassCcy="true"` for the investor's view and one `ccy="EUR" isFundCcy="true"` for the fund's reconciliation. There are **no** `GrossAmount` / `NetAmount` / `EntryLoad` / `ExitLoad` / `NavUsed` fields on `FlowType` . Entry loads and swing-pricing adjustments are operational concepts that the core schema does not model; producers that need to carry them use `<CustomAttributes>` on the share class or the regulatory-template cost fields of Chapter 8 (§7.9 returns to this).

`<TransactionID>` follows the same logic as `UniqueID` in Chapter 6 in the *conventional* sense: producers should make it unique within the delivery and stable across consecutive deliveries, so that consumers can reconcile. Unlike `UniqueID` on `Asset` , though, `TransactionID` is **not** declared as `xs:ID` in the schema — it is a plain `Text256Type` . This means the parser does not enforce uniqueness, and two `<Flow>` entries with the same `TransactionID` will schema-validate; the uniqueness discipline is purely a business convention. The recommended project-level convention is a deterministic string derived from the producer's order-management-system identifier, the event type, and the date — something like `EGF-TX-20260316-0024` . Chapter 13 will formalise this.

7.4 Subscriptions

The canonical `<Flow>` with `<TransactionType>SUB</TransactionType>` . Three subsections.

7.4.1 The Economic Model

A subscription is the operation in which an investor gives the fund cash and, in exchange, receives newly issued units of one of the fund's share classes. The economic flow is straightforward:

- The investor pays a cash amount into the share class's collection account.
- The share class computes how many units that cash amount buys at the relevant settlement NAV.

- New units are created; the class's total shares outstanding (which we saw in §5.12 as `ShareClass/TotalAssetValues/TotalAssetValue/SharesOutstanding`) increases by that count.
- The fund now has more cash to deploy — the portfolio manager will put the inflow to work in the next trading cycle by buying additional portfolio positions.

At the schema level, subscriptions are `<Flow>` entries inside the target share class's `<Flows>` container, with `<TransactionType>SUB</TransactionType>` as the discriminator. All of the common `FlowType` fields of §7.3 apply.

7.4.2 Pricing: When Is the NAV Struck

The critical conceptual question for subscriptions is *which NAV does the investor pay?* Three variants exist in the fund industry, but in the European UCITS world one of them dominates.

Forward pricing (the European standard): The investor places the order before a cut-off time (typically 13:00 CET for UCITS with daily valuation), and the NAV used to settle the order is the NAV that is computed at the *end* of the same trading day. At the moment the order is placed, the investor therefore does *not* know the price they are paying — they find out only after the markets close and the NAV has been computed. This is deliberate: forward pricing prevents "market timing", a practice in which an investor who already knows what the day's market movement will be exploits the stale NAV of the historic-pricing model to buy at yesterday's quote.

Historic pricing (rare, not UCITS-compatible): The investor buys at the last published NAV. This model is still used in a handful of AIFs and in some legacy structures, but it is effectively barred for UCITS by the regulation that followed the market-timing scandals of the early 2000s.

Market pricing (only for exchange-traded share classes): The investor buys the units on a stock exchange at the market quote, not at the fund's NAV. Creation and redemption by authorised participants happens in parallel through NAV-based processes, but ordinary retail investors in listed classes trade on the exchange at bid/ask prices that reflect, but are not identical to, the underlying NAV.

The Europa Growth Fund uses forward pricing on all three share classes. The cut-off is 13:00 CET on every Luxembourg trading day; orders received before the cut-off are settled at the NAV of the same day, and orders received after the cut-off are settled at the NAV of the next trading day. The consequence for the four date fields of §7.3 is concrete: `<TradeDate>` is the day the order was placed, `<AccountingDate>` equals `<TradeDate>` if the order was before the cut-off or equals `<TradeDate> + 1 business day` if it was after (the schema models the pricing-day decision through the accounting date, not through a dedicated field), and `<SettlementDate>` equals `<AccountingDate> + 2 business days` — T+2, the Luxembourg standard.

A note on what the schema does not model. FundsXML 4.2.8 does not carry the NAV that was actually applied to the subscription as an element on the `<Flow>`. The NAV is looked up indirectly by the consumer: the consumer reads the `<AccountingDate>` from the flow, then reads the NAV for that date from the share class's `<Prices>/<Price>` history (Chapter 5 §5.12.2). This decoupling is deliberate — the NAV history lives in one place, the transactions in another — and it keeps both halves small and composable. Similarly, entry loads, gross-versus-net breakdowns, and order-execution variants are **not** schema fields on `<Flow>`. §7.9 revisits why and explains what producers do when they need to carry them.

7.4.3 Subscription Example: A Retail Investor Subscribes to R-EUR-ACC

A complete XML example for a realistic subscription transaction. On Monday 16 March 2026 a retail investor subscribes 10,000 EUR to the R-EUR-ACC class of the Europa Growth Fund. The order is placed at 11:30 CET, comfortably before the 13:00 cut-off, so it is settled at the Monday evening NAV of 16 March 2026 — let us say 123.9504 EUR, representing a mid-March level between the February month-end and the 31 March figure of 124.5078 we saw in Chapter 5. The investor receives $10000 / 123.9504 = 80.6852$ units. The `<Flow>` entry lives inside `<ShareClass>/<Flows>` for the R-EUR-ACC class (ISIN LU2100000011).

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000011]/Flows -->
<Flow>
  <ActionCode>C</ActionCode>
  <TransactionID>EGF-TX-20260316-0024</TransactionID>
  <TradeDate>2026-03-16</TradeDate>
  <AccountingDate>2026-03-16</AccountingDate>
  <SettlementDate>2026-03-18</SettlementDate>
  <ValueDate>2026-03-16</ValueDate>
  <TransactionType>SUB</TransactionType>
  <Units>80.6852</Units>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">10000.00</Amount>
  </TotalValue>
  <Channel>Retail</Channel>
</Flow>
```

Reading field by field: the `<ActionCode>C</ActionCode>` marks the entry as a new (create) record — the schema demands it, and every fresh transaction in a delivery carries `C`. The `<TransactionID>` is the producer's unique key. The four date fields reflect forward pricing: same-day trade and accounting (Monday pre-cut-off), T+2 settlement on Wednesday, and an explicit `<ValueDate>` for completeness. The **mandatory** `<TransactionType>SUB</TransactionType>` identifies the direction. `<Units>` is positive — the schema convention, regardless of direction — and carries the 80.6852 units the investor receives. The mandatory `<TotalValue>` carries the 10,000 EUR amount, flagged as being in the fund's base currency. The optional `<Channel>Retail</Channel>` annotates the customer segment.

Two observations are worth making. First, note what is *not* in the listing: no `FlowType`, no `ShareClassISIN` (the class is identified by the parent element), no `Amount`, no `GrossAmount`, no `NetAmount`, no `EntryLoad`, no `NavUsed`, no `OrderExecutionType`. All of those would be schema-invalid as direct children of `<Flow>`. Second, the consistency check $10000.00 / \text{NavUsed} = \text{Units}$ still applies in spirit, but the consumer has to look up the NAV from a different container: it reads the NAV for `AccountingDate=2026-03-16` from `ShareClass/Prices/Price` and then verifies that $\text{Units} \times \text{NavPrice} \approx \text{TotalValue}$. The two halves of the check are in two different places, and that is by design.

7.5 Redemptions

The mirror operation of subscription, modelled as the second value of `<TransactionType>` — `RED` — in the same `FlowType` structure.

7.5.1 The Economic Model

A redemption is the operation in which an investor gives their units back to the fund and, in exchange, receives cash. The economic flow reverses the subscription:

- The investor submits a redemption order, typically for a specific number of units or for a specific cash amount.
- The share class computes how much cash those units are worth at the settlement NAV.
- The units are destroyed; the class's total shares outstanding decreases by that count.
- The fund pays out the amount and must source the cash for the payment, typically by selling portfolio positions in the next trading cycle.

At the schema level, a redemption is a `<Flow>` entry with `<TransactionType>RED</TransactionType>`. Every common `FlowType` field from §7.3 applies, with one important reminder: `<Units>` is still positive. The schema does not use the sign of `<Units>` to distinguish a redemption from a subscription — direction comes exclusively from `<TransactionType>`. A producer that writes a negative `<Units>` value on a redemption is fighting the schema's explicit convention.

7.5.2 Swing Pricing, Entry/Exit Loads, and the Concepts the Core Schema Does Not Model

A subtlety that distinguishes redemptions (and large subscriptions) from simple mirror arithmetic: when the fund has to redeem a large number of units at once, the payment produces *transaction costs* in the portfolio — the portfolio manager has to sell shares, and the sales themselves move the market slightly against the fund through the bid-ask spread and market impact. Without an adjustment, the *remaining* unit holders would bear the cost of the departing unit holders — a silent transfer of value from stayers to leavers that is known in the industry as "dilution".

Swing pricing is the standard mechanism to neutralise the dilution. When the net number of redemptions on a given trading day exceeds a threshold, the NAV used to settle *all* redemptions of that day is "swung" downward by a small amount — typically 10 to 50 basis points. The departing investors receive slightly less than the "normal" NAV, and the difference compensates for the transaction costs that will be incurred in the portfolio. Three swing-pricing variants exist in practice: **full swing** (applied to every transaction regardless of size), **partial swing** (applied only when net activity crosses a threshold), and **dual pricing** (separate bid and offer NAVs published permanently). The Europa Growth Fund operates partial swing with a 2% threshold.

The FundsXML 4.2.8 core schema does not model swing pricing, dilution levies, entry loads, or exit loads on `FlowType`. This is a deliberate architectural choice. The transaction container carries the *what happened* record — date, direction, unit count, total value — and leaves the pricing-adjustment details to other places in the document. Producers who need to carry these concepts have three options:

- **<CustomAttributes>** on the share class: Chapter 9 treats the **<CustomAttributes>** mechanism in detail, and for a share class with a swing-pricing regime a producer typically emits attributes such as **SwingPricingType** (with values *Full/Partial/Dual*), **SwingPricingThreshold** (a percentage), and **SwingPricingFactor** (basis points applied when the threshold is crossed). These attributes describe the *regime*, not individual swing events; the swing applied to a specific transaction is left implicit in the difference between **Flow/TotalValue** and **Units × Price** looked up from **ShareClass/Prices**.
- **The EMT regulatory templates** of Chapter 8 carry the full MiFID II cost disclosure, including entry and exit costs, ongoing charges, and performance fees. For regulatory consumers these are the authoritative source; for operational consumers the swing-pricing policy is a private contract between producer and distributor, carried through the custom attributes above.
- **Project-level convention.** Chapter 13 recommends specific **<CustomAttributes>** keys for swing pricing and entry-load carrying when producers and consumers agree on the convention.

There is therefore no XML example in this sub-section of a **FlowType** record that carries swing-pricing information directly — such an example would be schema-invalid. The operational concepts remain important, and Chapter 13 returns to them in the context of implementation projects; but readers who expect to find **<NavUsed>**, **<SwingAdjustment>**, **<NetNavUsed>**, **<EntryLoad>**, or **<ExitLoad>** on a FundsXML flow will not find them, because they are not there.

7.5.3 Redemption Gates and Deferred Redemptions

An extreme case: when a fund receives more redemption orders than it can serve in a single trading cycle (typically because the underlying positions are illiquid and cannot be sold fast enough), the fund can **hold back** redemptions. Two mechanisms exist. **Redemption gates** execute only a fraction of the orders on the current day and defer the rest to the next day (or over several days). **Suspensions** freeze redemptions entirely until the liquidity situation improves. Both mechanisms are provided for in UCITS prospectuses but are practically activated only in crises — the 2008 financial crisis, the March 2020 COVID shock, the 2022 UK gilt-market turbulence that hit LDI funds.

FundsXML has no dedicated fields for gates or suspensions either. Partial executions are modelled as separate **<Flow>** records — a redemption for the executed units today, a second redemption for the remaining units tomorrow, both tied together by a common **<TransactionID>** prefix or by referring to each other through **<OriginalTransactionID>** in the same spirit as the cancellation mechanism of §7.3. The Europa Growth Fund has never activated a gate, and the example block in §7.10 will not show one.

7.5.4 Redemption Example

An institutional redemption from the I-EUR-DIST class: on 20 March 2026 a pension fund submits an order to redeem 15,000 units. At the Friday evening NAV of that day — 107.8421 EUR, a mid-March value — the redemption is worth approximately 1.62 million EUR. No swing (the amount is well below the 2% threshold). Settlement T+2 puts the cash payment on 24 March. The **<Flow>** entry lives inside **<ShareClass>/<Flows>** for the I-EUR-DIST class (ISIN LU2100000037).

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000037]/Flows -->
<Flow>
  <ActionCode>C</ActionCode>
```

```

<TransactionID>EGF-TX-20260320-0011</TransactionID>
<TradeDate>2026-03-20</TradeDate>
<AccountingDate>2026-03-20</AccountingDate>
<SettlementDate>2026-03-24</SettlementDate>
<ValueDate>2026-03-20</ValueDate>
<TransactionType>RED</TransactionType>
<Units>15000.0000</Units>
<TotalValue>
  <Amount ccy="EUR" isFundCcy="true">1617631.50</Amount>
</TotalValue>
<Channel>Institutional</Channel>
</Flow>

```

Reading field by field, focusing on the differences from §7.4.3: the single mandatory change is `<TransactionType>RED</TransactionType>` — that one field is the entire difference between a subscription and a redemption at the schema level. `<Units>` is **positive** (15000.0000), following the schema's explicit convention, even though the economic direction is outflow. The `<Channel>Institutional</Channel>` annotates the customer segment differently from the retail subscription of §7.4.3. Everything else — the four date fields, the `<ActionCode>`, the `<TotalValue>` — follows the same pattern as the subscription example, which is the whole point: subscriptions and redemptions are the same schema shape distinguished by a single enum value.

7.6 Switches — Two Linked Flow Entries

The third transaction type conceptually, but **not** a third schema type. This is the single biggest surprise of the chapter for anyone coming from a textbook fund-data background.

7.6.1 The Schema Has No "Switch" Transaction Type

A switch is, economically, the combination of two events: a redemption from one share class (the *source*) and a simultaneous subscription to another share class (the *target*). The distinguishing feature is that both legs are settled at the *same* accounting date, at the *same* trade timestamp, and the two are economically linked — the proceeds of the redemption flow directly into the subscription, without the investor receiving cash in between.

The FundsXML 4.2.8 schema has no dedicated *Switch* transaction type. The `<TransactionType>` enumeration on `FlowType` has only two values, `SUB` and `RED`, and there is no third value for switches. Consequently, a switch is modelled as **two separate `<Flow>` entries**:

- one `<Flow>` with `<TransactionType>RED</TransactionType>` inside the source class's `<Flows>` container;
- one `<Flow>` with `<TransactionType>SUB</TransactionType>` inside the target class's `<Flows>` container.

The two entries are linked by convention, not by a first-class schema relationship. The two most common linking conventions are:

- **Shared `<TransactionID>` prefix:** both legs carry a `<TransactionID>` whose prefix identifies the switch (e.g. EGF-SW-20260312-0007-SRC and EGF-SW-20260312-0007-TGT) so that a consumer can pair them by string match.
- **`<TransactionSubtype>` annotation:** both legs carry a `<TransactionSubtype>` value starting with `switch` (the schema element is a free-text `xs:string` , so any convention is allowed) together with a reference to the paired leg's `<TransactionID>` . The schema documentation lists *restructuring plan* and similar free-text conventions as typical values.

Most producers use both conventions simultaneously. A consumer that wants to reconstruct switches walks every `<Flow>` across every share class, groups pairs by the prefix or by the subtype reference, and reports the combined event to its downstream pipeline as a single conceptual switch. A consumer that does *not* know to look for the pairing reads the two legs as an independent redemption and an independent subscription — and that is an acceptable reading for most use cases, because the two legs net to zero at the fund level. The switch interpretation matters only for consumers that track investor-level holdings across classes, and those consumers know to look for the pairing.

This two-record model is a genuine surprise for anyone coming from a format that models switches as first-class transactions. The schema's choice is deliberate: it avoids introducing a three-valued `<TransactionType>` that would need its own rules for quantity and cash-flow direction, and it lets every `<Flow>` continue to be a self-contained record that can be processed independently. The cost is that switch reconciliation is a consumer-side concern, and the quality of the pairing depends on the producer's naming discipline.

7.6.2 Intra-Fund Versus Inter-Fund Switches

Two variants exist, and they differ in operational meaning as well as in schema location.

An **intra-fund switch** moves an investor's holding from one share class to another of the *same* fund. Example: a retail investor in the R-EUR-ACC class of the Europa Growth Fund crosses the institutional threshold through portfolio growth and switches to the I-EUR-DIST class to benefit from the lower institutional fee schedule. The switch is economically a non-event for the fund itself — the underlying portfolio does not change; only the share-class attribution of one investor's stake shifts. `SharesOutstanding` of R-EUR-ACC decreases, `SharesOutstanding` of I-EUR-DIST increases, and the fund's total net assets remain unchanged.

In the schema, both legs of an intra-fund switch live in the **same fund's** `<SingleFund>/<ShareClasses>` block — one in the source class's `<Flows>` , one in the target class's `<Flows>` . Both are in the same delivery and both refer to classes visible in the same document.

These switches are *cost-free* for the fund because they do not trigger any portfolio activity, and most asset managers therefore charge no switch fee on intra-fund moves. They are nevertheless transactions that must be reported, because they change the distribution of shares outstanding across the classes, and consumers that track per-class metrics need to see both legs.

An **inter-fund switch** moves an investor's holding between share classes of *different* funds that belong to the same umbrella. Example: an investor in the Europa Growth Fund wants to move to another sub-fund of the same SICAV — a hypothetical *Europa Asset Management Investments — Emerging Markets Sub-Fund*, for instance. The umbrella structure introduced in Chapter 5.14 permits this kind of switch, and many distributors offer it as a convenience feature that avoids the tax friction of a full market-exit-and-re-entry.

Inter-fund switches are more complex because they touch the portfolios of *both* funds: the source fund must raise cash (selling positions), the target fund must deploy cash (buying positions). In the schema, the two `<Flow>` legs live inside *different* `<Fund>` elements — one inside the source fund's `<SingleFund>/<ShareClasses>/<ShareClass>/<Flows>`, one inside the target fund's. Both affected funds report their own leg in their own delivery (or in different segments of the same multi-fund delivery), and a consumer that reconstructs the full switch needs to see both funds' data.

The Europa Growth Fund, as we established in Chapter 5.14, is shown as a standalone fund in the running example rather than as a sub-fund of a larger umbrella. Inter-fund switches are therefore not represented in the chapter's complete example; Chapter 13 will return to them when it treats umbrella implementation projects.

7.6.3 Switch Example

A small intra-fund switch: on 12 March 2026 an investor moves a 50,000 EUR stake from the R-EUR-ACC class to the I-EUR-DIST class. The amount is in reality below the institutional minimum of 1,000,000 EUR and the switch would be rejected by a well-configured order-management system, but for the purpose of illustrating the structure we let it through. The source NAV is 123.7021 EUR (a 12 March value for R-EUR-ACC), the target NAV is 107.7413 EUR (the same day for I-EUR-DIST). The investor gives up $50000 / 123.7021 = 404.2156$ units of R-EUR-ACC and receives $50000 / 107.7413 = 464.0882$ units of I-EUR-DIST.

Leg 1 — source redemption, inside the R-EUR-ACC class's `<Flows>` :

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000011]/Flows -->
<Flow>
  <ActionCode>C</ActionCode>
  <TransactionID>EGF-SW-20260312-0007-SRC</TransactionID>
  <TradeDate>2026-03-12</TradeDate>
  <AccountingDate>2026-03-12</AccountingDate>
  <SettlementDate>2026-03-16</SettlementDate>
  <ValueDate>2026-03-12</ValueDate>
  <TransactionType>RED</TransactionType>
  <TransactionSubtype>switch – source leg; paired with EGF-SW-20260312-0007-TGT on
LU2100000037</TransactionSubtype>
  <Units>404.2156</Units>
  <TotalValue>
    <Amount ccy="EUR" isFundCcy="true">50000.00</Amount>
  </TotalValue>
  <Channel>Intra-fund switch</Channel>
</Flow>
```

Leg 2 — target subscription, inside the I-EUR-DIST class's `<Flows>` :

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000037]/Flows -->
<Flow>
  <ActionCode>C</ActionCode>
  <TransactionID>EGF-SW-20260312-0007-TGT</TransactionID>
  <TradeDate>2026-03-12</TradeDate>
  <AccountingDate>2026-03-12</AccountingDate>
```

```

<SettlementDate>2026-03-16</SettlementDate>
<ValueDate>2026-03-12</ValueDate>
<TransactionType>SUB</TransactionType>
<TransactionSubtype>switch – target leg; paired with EGF-SW-20260312-0007-SRC on
LU2100000011</TransactionSubtype>
<Units>464.0882</Units>
<TotalValue>
  <Amount ccy="EUR" isFundCcy="true">50000.00</Amount>
</TotalValue>
<Channel>Intra-fund switch</Channel>
</Flow>

```

Four points are worth noting about this pair.

First, the two legs live in **different parent elements** — one inside the R-EUR-ACC `<ShareClass>`, one inside the I-EUR-DIST `<ShareClass>`. They are not siblings. A consumer that scans `<Flow>` entries in document order and looks for pairs has to walk across share classes.

Second, the two `<Units>` values are not equal, because the two classes have different NAVs — the investor gives up 404.2156 units of R-EUR-ACC and receives 464.0882 units of I-EUR-DIST, both corresponding to a 50,000 EUR value. The unit count changes, but the economic value does not. Both `<Units>` values are **positive**, following the schema convention, and the direction is carried through `<TransactionType>` (RED on the source, SUB on the target).

Third, both legs carry identical `<TotalValue>` — the 50,000 EUR common amount that ties the two legs together economically. A consumer that wants to compute the net flow for the fund sees the source leg as an outflow (because `<TransactionType>RED</TransactionType>`) and the target leg as an inflow (because `<TransactionType>SUB</TransactionType>`); the two cancel to zero at the fund level. A consumer that does not recognise the switch pairing still gets the right answer — because the +50,000 and -50,000 net to zero — but displays the move as two separate transactions on the investor's statement instead of as one.

Fourth, the `<TransactionSubtype>` free-text annotation starts with the word *switch* and names the paired leg's `<TransactionID>`. This is a project-level convention rather than a schema rule, but it is the standard way to make the pairing discoverable to a consumer that wants to reconstruct switches. Chapter 13 formalises the convention for projects that care about investor-level reconciliation.

7.7 Distributions Paid to Investors

The fourth event type — and one that, for the Europa Growth Fund on 31 March 2026, does not apply. The distribution for the I-EUR-DIST class is an annual event in mid-May, and no distribution event belongs to the March reporting period. This section treats the mechanics and the fields nonetheless, because other funds use distributions constantly and because the Europa Growth Fund will emit a distribution event in its May 2026 delivery.

7.7.1 The Schema Container

Distributions live in `ShareClass/Distributions/Distribution`, schema type `DistributionType` (XSD §7197). The container is attached to the individual distributing share class — in the Europa Growth Fund's case, to the I-EUR-DIST class only — and lists the distribution events that have been declared for that class. A delivery that carries no distribution event for the reporting period either omits the `<Distributions>` container entirely or emits it empty; both forms are schema-valid.

The container is **not** period-scoped the way `<Flows>` and `<Earnings>` are; there are no `@from` / `@to` attributes on `<Distributions>`. A delivery carries the distribution events whose announcement or ex-date falls inside the reporting period, but the scoping is purely by content, not by a container attribute.

7.7.2 The Mechanics of a Distribution and the Four Dates

A distribution is the operation in which a distributing share class pays a portion of its accumulated income out to its investors. The mechanics run through four distinct dates, all of which the schema models:

- **Announcement date** (`<AnnouncementDate>`, optional) — the day the fund's board resolves the distribution and publishes its amount. The schema calls this `AnnouncementDate` rather than the legacy industry term *declaration date*, and a producer writing `<DeclarationDate>` will fail validation.
- **Record date** (`<RecordDate>`, optional) — the day the list of distribution recipients is formally recorded. Typically one or two trading days after the ex-date.
- **Ex-date** (`<ExDate>`, **mandatory**) — the day the class's units trade "without entitlement to the distribution". Any investor holding units at the ex-date receives the distribution; anyone who buys units after the ex-date does not. The class's NAV drops on the ex-date by the per-unit distribution amount, because the class has booked the distribution as a liability to its investors.
- **Payment date** (`<PaymentDate>`, **mandatory**) — the day the cash actually flows to the investors' accounts. Several weeks after the ex-date in most jurisdictions.

The order in the schema sequence is unusual: `AnnouncementDate?`, `RecordDate?`, `ExDate`, `PaymentDate`. Note that `RecordDate` comes before `ExDate` in XML element order, even though operationally it happens after. A producer must emit the elements in schema order regardless of the operational sequence of events.

7.7.3 The Amount Structure

The distribution amount is carried through **two** mandatory amount blocks that mirror each other: `<GrossDividendAmount>` and `<NetDividendAmount>`. Each is a small structured element with two children:

- `<Total>` (optional, `PositiveAmountType`) — the total payout flowing out of the fund.
- `<PerShare>` (mandatory, `PositiveAmountType`) — the payout per unit of the share class.

Both amount blocks use `PositiveAmountType`, which is `FundAmountType` restricted to non-negative values. The element names are `DividendAmount` even for distributions that are not formally dividends — the schema uses *dividend* as a generic term for "payout to investors".

There is no `<DistributionSubType>` element. The schema does not distinguish Interim / Final / CapitalGains / ReturnOfCapital at the `DistributionType` level. A producer that needs to carry the distinction does so either through the free-text `<Comments>` field at the bottom of `DistributionType` or through `<CustomAttributes>` on the share class. In practice the subtype is a tax-reporting concern rather than a schema concern, and the distinction is carried in the regulatory reporting of Chapter 8.

There is no `<SharesAtRecord>` element. The shares outstanding at the record date are not a field of `DistributionType`. A consumer that needs the figure reads it from the share class's `<TotalAssetValues>/<SharesOutstanding>` as of the record date, or computes `Total / PerShare` from the two amount children.

7.7.4 Other Mandatory Fields

On top of the four dates and the two amount blocks, `DistributionType` carries three more mandatory children that a producer must populate on every entry:

- `<ActionCode>` — enum C / D / M, as on every other dynamic record in the schema.
- `<DividendStatus>` — enum ESTIMATED or OFFICIAL. ESTIMATED is used for distributions whose amount has been announced but not yet formally resolved; OFFICIAL for distributions whose amount is final. A delivery between announcement and formal resolution might carry the same distribution first with ESTIMATED and then, in a later delivery, with OFFICIAL.
- `<PaymentCurrency>` — ISO 4217 code. Usually matches the share class's currency but can differ for classes that pay out in a different currency than they are denominated in (rare).

Seven mandatory children in total, in this order: `ActionCode`, `DividendStatus`, `ExDate`, `PaymentDate`, `PaymentCurrency`, `GrossDividendAmount`, `NetDividendAmount`.

7.7.5 The Europa Growth Fund's Situation and a Hypothetical Example

Honest disclosure to the reader: **the 31 March 2026 delivery of the Europa Growth Fund contains no distribution events**. The reason is straightforward: the I-EUR-DIST class pays its annual distribution in mid-May, with an announcement date in early May, an ex-date around 15 May, a record date around 17 May, and a payment date around 27 May. The March delivery sees none of this. The complete example in §7.10 will respect this fact and will not contain a distribution entry, rather than fabricating one for pedagogical convenience.

For the purpose of illustrating the *structure*, however, this subsection shows the hypothetical May distribution as it would appear in the May 2026 delivery of the fund. The assumed values are: announcement date 5 May, record date 17 May, ex-date 15 May, payment date 27 May, gross per share 3.80 EUR, net per share 3.50 EUR (after 7.9% withholding at the fund level), shares outstanding 2,350,000 at record.

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000037]/Distributions -->
<Distribution>
  <ActionCode>C</ActionCode>
  <DividendStatus>OFFICIAL</DividendStatus>
```



```

<AnnouncementDate>2026-05-05</AnnouncementDate>
<RecordDate>2026-05-17</RecordDate>
<ExDate>2026-05-15</ExDate>
<PaymentDate>2026-05-27</PaymentDate>
<PaymentCurrency>EUR</PaymentCurrency>
<GrossDividendAmount>
  <Total>
    <Amount ccy="EUR" isFundCcy="true">8930000.00</Amount>
  </Total>
  <PerShare>
    <Amount ccy="EUR" isShareClassCcy="true">3.80</Amount>
  </PerShare>
</GrossDividendAmount>
<NetDividendAmount>
  <Total>
    <Amount ccy="EUR" isFundCcy="true">8225000.00</Amount>
  </Total>
  <PerShare>
    <Amount ccy="EUR" isShareClassCcy="true">3.50</Amount>
  </PerShare>
</NetDividendAmount>
<DistributionFlag>true</DistributionFlag>
</Distribution>

```

Reading field by field: `<ActionCode>C</ActionCode>` marks the entry as a create record. `<DividendStatus>OFFICIAL</DividendStatus>` signals that the distribution amount has been formally resolved by the board. The four dates — `AnnouncementDate`, `RecordDate`, `ExDate`, `PaymentDate` — are listed in schema order (not in operational order). `<PaymentCurrency>EUR</PaymentCurrency>` matches the I-EUR-DIST class's currency. The two mandatory amount blocks, `<GrossDividendAmount>` and `<NetDividendAmount>`, each carry the total payout and the per-share figure; the net is smaller because of a 7.9% fund-level withholding. `<DistributionFlag>true</DistributionFlag>` is an optional boolean that explicitly confirms the distribution will actually be distributed (as opposed to reinvested as an accrual).

One important corollary that the schema models indirectly: the NAV of the I-EUR-DIST class *drops* on the ex-date of 15 May 2026 by approximately 3.80 EUR per unit. If the NAV on 14 May was, say, 111.20 EUR, the NAV on 15 May would be approximately 107.40 EUR (111.20 minus 3.80), ignoring any other market movement between the two valuation points. The NAV time series carried in `ShareClass/Prices/Price` (Chapter 5 §5.12.2) shows this as a discrete downward jump — exactly the sawtooth pattern that distinguishes distributing classes from their accumulating counterparts, as §5.10.1 explained. The drop is not a loss; it is the mechanical consequence of moving accumulated income from "inside the NAV" to "inside the investors' cash accounts".

7.8 Income Received from the Portfolio

The fifth event type — and the counterpart to §7.7: money that flows *into* the fund from its portfolio holdings, rather than out from the fund to investors. The schema container is different, the schema type is different, and the wrapper element lives one level up — at the fund-level portfolio rather than inside a share class.

7.8.1 The Schema Container and the Fund-Portfolio-Not-Share-Class Rule

Income events live in `Portfolio/Earnings/Earning`, schema type `EarningType` (XSD §7556). The `<Earnings>` container is a child of `Portfolio` and appears inside `Fund/FundDynamicData/Portfolios/Portfolio` — the same fund-level portfolio where the positions and transactions of Chapter 6 live. The container carries two optional attributes, `@from` and `@to`, that bracket the reporting period.

The critical difference from §7.7 is the level of attachment. **Income events are attached to the fund's portfolio, not to any share class.** When the fund receives a dividend from a portfolio holding, the event is a property of the fund's asset pool, not of a specific class. All three share classes benefit pro-rata from the income through their share of the fund's total net assets; that pro-rata split is implicit in the valuation-day NAV calculation and is not re-recorded in the transaction data.

The distinction from §7.7 is important enough to repeat: **distributions in `ShareClass/Distributions` flow out of a share class to its investors; earnings in `Portfolio/Earnings` flow into the fund from its portfolio positions.** The two are economic opposites, live in different containers, and use different schema types. A producer that mislabels an income event as a distribution causes a pair of reconciliation failures: the fund appears to lose cash that it should have gained, and the distribution appears to come from an imaginary source.

7.8.2 The `EarningKind` Enumeration

Every `<Earning>` carries a mandatory `<EarningKind>` element drawn from a fixed six-value enumeration:

- **Coupon** — a coupon payment on a bond held by the fund. The bond pays its periodic coupon (annually, semi-annually, or quarterly depending on the coupon frequency from §6.7.1) and the fund receives the corresponding cash. Coupons are *contractually fixed* in the bond prospectus, not decided each year by the issuer. The `<InterestClaimGross>` from §6.7.2 resets to zero at the coupon payment date, and the accumulated amount flows as cash into the fund.
- **Dividend** — a dividend on an equity held by the fund. The most common case in an equity fund like the Europa Growth Fund. The workflow follows the usual ex-date / record / payment pattern, but from the fund's perspective the ex-date is the day the claim arises (booked as a receivable against the issuer) and the payment date is the day the cash arrives. The gap between the two is typically several weeks.
- **Fund distribution** — when the fund holds a position in another fund (§6.8), and that fund distributes to its holders, the received distribution is an income event from our fund's perspective.
- **Interest deposit/giro** — interest on cash balances. The custodian bank pays the fund periodically at the agreed rate on its balances (or charges negative interest, depending on the market environment). Most commonly settled monthly.
- **Interest swap** — interest payments arising from a swap contract's leg.

- **Other** — the catch-all for events that do not fit the first five.

Note the exact spelling of each value — the schema is case-sensitive and space-sensitive. `Interest deposit/giro` with its lowercase "deposit" and embedded slash is the real spelling; a producer that writes `InterestDeposit` or `Interest Deposit` will fail validation.

7.8.3 The Common `EarningType` Fields and the Typed Sub-Block

`EarningType` has two layers. The **common** layer carries fields that apply to every kind of earning; the **typed** layer is an optional choice of three sub-elements — `<Coupon>`, `<Dividend>`, `<FundDistribution>` — that carries kind-specific breakdowns.

The mandatory common fields, in schema order, are:

- **`<EarningID>`** — a `Text256Type` unique identifier for the earning record. Analogous to `<TransactionID>` on `FlowType` and the same conventions apply (unique within delivery, stable across deliveries, not declared as `xs:ID`).
- **`<EarningKind>`** — the six-value enumeration from §7.8.2.
- **`<EntryDate>`** — the date the earning was booked into the portfolio. Analogous to `<AccountingDate>` on `FlowType`.
- **`<EntryCurrency>`** — the currency in which the earning is booked in the fund's ledger. Usually the fund's base currency; can differ for funds that keep their books in multiple currencies.
- **`<EntryValue>`** — a `FundAmountType` carrying the consolidated value of the earning in the fund's base currency (and optionally also in the income currency).

Optional common fields include `<CancellationFlag>` and `<OriginalEarningID>` for reversal records, `<AssetUniqueID>` (an `xs:IDREF` pointing at the asset in `AssetMasterData` — the same IDREF mechanism as positions use), `<Identifiers>` for an inline copy of the instrument codes, `<ClosingDate>` (the trade date — earlier than `EntryDate` when the booking lags the trade), `<ValutaDate>` (the effective date for interest-accrual purposes), `<IncomeCurrency>` (the currency the issuer paid in, when different from `EntryCurrency`), `<Interests>` (interest value where applicable), `<NominalValueOrUnits>` (the shares or nominal amount the earning was based on), `<ReferenceNumber>`, `<AccountNumber>`, `<PostingText>`, and `<FinallySettled>`.

The **typed sub-block** at the end of `EarningType` is an optional choice of three inline complex types:

- **`<Dividend>`** — carries `<DividendPerUnitValue>`, `<DividendGrossValue>`, `<DividendNetValue>`, `<WithholdingTaxValue>`, `<ExDay>`, `<PayDay>`, `<WithholdingTaxQuota>`. All optional, but populated on every real dividend event.
- **`<Coupon>`** — carries `<UnitInterestsGrossValue>`, `<UnitInterestsNetValue>`, `<CapitalYieldsTaxValue>`, `<EUWithholdingTaxValue>`.
- **`<FundDistribution>`** — carries `<PayoutPerUnitValue>`, `<PayoutGrossValue>`, `<CapitalYieldsTaxUnitValue>`, `<CapitalYieldsTaxOverallValue>`, `<EUWithholdingTaxUnitValue>`, `<EUWithholdingTaxOverallValue>`, `<PayoutIdenticalEarningsValue>`, `<ExDay>`.

Exactly one of the three may be present (the choice is optional), and its name matches the `<EarningKind>` value on the common layer: an `EarningKind=Dividend` entry carries a `<Dividend>` sub-block, a `Coupon` entry carries a `<Coupon>` sub-block, and so on. The three cash enumerations (`Interest deposit/giro`, `Interest swap`, `Other`) do not have their own sub-block and populate only the common layer.

7.8.4 Withholding Tax

A tax detail that sets income events apart from the other event types: **foreign dividends are typically taxed at source**. A Luxembourg-domiciled fund receiving dividends from a German share sees approximately 26.375% German withholding tax deducted before the net amount arrives in Luxembourg. Depending on the double-taxation treaty between the two countries and on the fund's legal structure, a portion of the withheld amount may later be reclaimable, but at the moment of the payment the fund sees only the net.

FundsXML models withholding tax inside the typed sub-block. For a dividend event, the relevant fields are `<DividendGrossValue>`, `<DividendNetValue>`, `<WithholdingTaxValue>`, and `<WithholdingTaxQuota>` — all inside `<Earning>/<Dividend>`. Consumers that compute fund performance use `<DividendGrossValue>`; consumers that reconcile cash balances use `<DividendNetValue>`. Both are authoritative for their respective use cases, and both should be populated on every dividend from a taxable source. `<WithholdingTaxValue>` is the absolute amount kept at source, and `<WithholdingTaxQuota>` is the rate as a percentage (in the schema's standard human-readable per-cent convention: 26.375 means 26.375%, not 0.26375).

The common-layer `<EntryValue>` typically carries the net amount — the amount that actually arrived in the fund's ledger. A producer that wants to reconcile cash balances reads `EntryValue`; a producer that wants the gross figure reads `Dividend/DividendGrossValue`. The two are both populated, intentionally, and the consumer picks whichever is appropriate.

7.8.5 Income Example: A BNP Paribas Interim Dividend

An example of an income event drawn from the actual portfolio of the Europa Growth Fund. BNP Paribas (ISIN FR0000131104, 110,000 shares in the portfolio per Chapter 6) pays an interim dividend in Q1, with an ex-date of 20 March 2026 and a payment date of 24 March 2026. The interim amount is 1.50 EUR per share. The gross cash flow to the fund is $110,000 \times 1.50 = 165,000.00$ EUR. France applies a 25% withholding tax on dividends paid to foreign recipients under the France-Luxembourg tax treaty, so the fund receives $165,000.00 \times 0.75 = 123,750.00$ EUR net.

```
<!-- Inside Fund/FundDynamicData/Portfolios/Portfolio/Earnings -->
<Earning>
  <EarningID>EGF-EARN-20260320-0012</EarningID>
  <AssetUniqueID>FR0000131104</AssetUniqueID>
  <EarningKind>Dividend</EarningKind>
  <ClosingDate>2026-03-20</ClosingDate>
  <EntryDate>2026-03-24</EntryDate>
  <ValutaDate>2026-03-24</ValutaDate>
  <EntryCurrency>EUR</EntryCurrency>
  <IncomeCurrency>EUR</IncomeCurrency>
  <EntryValue>
```

```

    <Amount ccy="EUR" isFundCcy="true">123750.00</Amount>
  </EntryValue>
  <NominalValueOrUnits>110000</NominalValueOrUnits>
  <PostingText>BNP Paribas Q1 interim dividend, net of 25% French withholding
tax</PostingText>
  <Dividend>
    <DividendPerUnitValue>
      <Amount ccy="EUR">1.50</Amount>
    </DividendPerUnitValue>
    <DividendGrossValue>
      <Amount ccy="EUR" isFundCcy="true">165000.00</Amount>
    </DividendGrossValue>
    <DividendNetValue>
      <Amount ccy="EUR" isFundCcy="true">123750.00</Amount>
    </DividendNetValue>
    <WithholdingTaxValue>
      <Amount ccy="EUR">41250.00</Amount>
    </WithholdingTaxValue>
    <ExDay>2026-03-20</ExDay>
    <PayDay>2026-03-24</PayDay>
    <WithholdingTaxQuota>25.00</WithholdingTaxQuota>
  </Dividend>
</Earning>

```

Reading field by field: the `<EarningID>` is the producer's unique key for the earning. The `<AssetUniqueID>` is the back-pointer to the asset entry in `AssetMasterData` (Chapter 6) through an `xs:IDREF` — the consumer sees `FR0000131104` and knows immediately that the event concerns the BNP Paribas position. `<EarningKind>Dividend</EarningKind>` selects the `<Dividend>` sub-block at the bottom.

The three date fields on the common layer — `<ClosingDate>`, `<EntryDate>`, `<ValutaDate>` — tell different parts of the story. `ClosingDate=2026-03-20` is the dividend's ex-date, the day the claim arose. `EntryDate=2026-03-24` is the day the cash was booked into the fund's ledger (four days later, once the payment actually arrived). `ValutaDate=2026-03-24` is the day the amount was effective on the cash account for interest-accrual purposes. For a simple dividend received into a current account, `EntryDate` and `ValutaDate` are usually the same day.

`<EntryCurrency>EUR</EntryCurrency>` names the ledger currency; `<IncomeCurrency>EUR</IncomeCurrency>` names the currency the issuer paid in. For a French dividend paid to a euro-base fund, both are EUR; for a British dividend paid to the same fund, `IncomeCurrency` would be `GBP` and `EntryCurrency` would still be `EUR` (reflecting the custodian's FX conversion). `<EntryValue>` carries the consolidated net amount in the fund's base currency; `<NominalValueOrUnits>110000</NominalValueOrUnits>` records the 110,000 shares the fund held at the ex-date — and the schema annotation deliberately uses the *nominal or units* naming to cover both equities (units) and bonds (nominal).

The `<Dividend>` sub-block at the bottom carries the kind-specific breakdown. `DividendPerUnitValue=1.50` EUR, `DividendGrossValue=165000.00` EUR, `DividendNetValue=123750.00` EUR, `WithholdingTaxValue=41250.00` EUR, `ExDay=2026-03-20`, `PayDay=2026-03-24`, `WithholdingTaxQuota=25.00`. Note the duplication: `ExDay` in the sub-block mirrors `ClosingDate` on the

common layer, and `PayDay` mirrors `EntryDate`. The duplication is deliberate — the sub-block names come from the dividend-specific vocabulary, the common-layer names come from the generic-earning vocabulary, and both are populated so that consumers reading either layer get a consistent picture.

One operational note: the `<NominalValueOrUnits>` field on the income event is the number of shares that *were* held at the record date of the dividend, not the number held on the delivery's valuation date. If the fund had bought additional BNP Paribas shares between the dividend's record date and the valuation date, the position from Chapter 6 would show the higher current count, while the earning event's `NominalValueOrUnits` would record the lower count that actually earned the dividend. The two figures therefore legitimately differ, and consumers that reconcile dividend receipts against positions need to use the record-date shares rather than the current shares.

7.9 Order Execution, Cut-offs, and the Operational Concepts the Core Schema Does Not Model

The concluding structural section of the chapter, and conceptually distinct from the preceding sections: it treats a set of operational concepts — forward pricing, cut-off times, order-execution variants, in-kind creation, market-traded execution — that **every** UCITS operates with, but that the FundsXML 4.2.8 core schema does not model on `<Flow>`. Producers that need to carry these concepts use mechanisms outside the core transaction containers, and this section walks through what exists, what doesn't, and where the operational data actually lives.

7.9.1 What the Schema Does and Does Not Carry on `<Flow>`

We said this in §7.4.2 and §7.5.2, but it deserves one more explicit statement: the core `FlowType` has no element called `OrderExecutionType`, no element called `NavUsed`, no element called `NetNavUsed`, and no element called `SwingAdjustment`. There are no enumeration values `AtNAV`, `ForwardDated`, `HistoricPriced`, `Market`, `Negotiated`, or `InKind` anywhere in the schema. A producer that writes any of these elements or values will fail schema validation.

The operational concepts these labels describe are real — forward pricing, cut-off times, dilution levies, in-kind creation, market-priced ETF trades, negotiated institutional blocks — and they matter every day in European fund distribution. The schema's position is that the `<Flow>` record captures the *outcome* of these mechanisms (the direction, the unit count, the total value, the dates) and not the *process* that produced them. If a consumer needs to know whether a specific trade was executed at NAV or on an exchange, that information lives somewhere else in the delivery or in a separate file entirely.

7.9.2 Cut-Off Times and the Forward-Pricing Model

The forward-pricing mechanism depends on a critical concept: the **cut-off time**, the moment in the trading day by which an order must be received in order to be settled at the NAV of *that* day. Orders received after the cut-off are rolled over to the *next* trading day.

The cut-off is understood at two levels in practice:

- The **fund cut-off** — specified in the prospectus, typically 12:00 or 13:00 in the time zone of the fund's domicile. For the Europa Growth Fund, this is 13:00 CET or CEST on Luxembourg trading days.
- The **distributor cut-off** — distributors typically set their own cut-offs a few hours *before* the fund cut-off, to give themselves enough time to aggregate and forward the collected orders to the fund. A German distributor might cut off at 10:30 CET, a French distributor at 11:00 CET.

The consequence: a subscription that reaches the distributor at 10:00 CET makes both the distributor cut-off and the fund cut-off and is settled at the same-day NAV. A subscription that reaches the distributor at 12:00 CET misses the distributor cut-off, even though it would have comfortably made the fund cut-off, and is rolled over to the next trading day's NAV. The distributor effectively adds a one-day delay.

How the schema captures the result. FundsXML captures the cut-off decision through the relationship between `<TradeDate>` and `<AccountingDate>` on `<Flow>`. For an order that made the cut-off, both dates are the same day. For an order that missed the distributor cut-off, `<TradeDate>` is the day the investor submitted the order and `<AccountingDate>` is the next trading day whose NAV was applied. A consumer that wants to detect "late" orders reads the difference between the two dates. The schema does not label the order as late or on-time — the dates speak for themselves.

The prospectus cut-off time itself is not carried in the core schema on a per-flow basis. Producers that want to transport cut-off metadata for each share class do so through `<CustomAttributes>` on the share class (Chapter 9), with attribute keys such as `CutOffTime` (a time value like 13:00) and `CutOffTimezone` (Europe/Luxembourg). These attributes describe the *regime*; individual `<Flow>` records do not repeat them.

7.9.3 Market-Priced Trades, In-Kind Creation, and Exchange-Traded Share Classes

For exchange-traded share classes — UCITS whose units trade on a public market in addition to the NAV-based primary-market channel — the investor-facing trades are executed at the exchange quote, not at the fund's NAV. The creation and redemption process with authorised participants happens in parallel through a NAV-based primary-market channel, but ordinary retail investors in listed classes trade on the exchange at bid/ask prices that reflect, but are not identical to, the underlying NAV.

The FundsXML 4.2.8 core schema does not model exchange-traded flows on `<Flow>`. A producer running an ETF-style share class has two options:

- **Do not emit `<Flow>` records for exchange trades at all.** The exchange-driven creations and redemptions by authorised participants are the only events that meaningfully change the fund's shares outstanding and cash balance, and those events are processed NAV-based through the primary market. Secondary-market exchange trades between investors do not affect the fund's balance sheet and are not reported to the fund.
- **Use `<CustomAttributes>` on the individual `<Flow>` to annotate the execution channel if the producer's business logic needs it.**

For in-kind creation and redemption — where a large institutional subscriber delivers a basket of securities to the fund in exchange for units, rather than cash — the schema approach is to emit a `<Flow>` with `<TransactionType>SUB</TransactionType>` and the `<TotalValue>` expressed as the market value of the delivered basket. The basket itself is not listed on the `<Flow>`; the positions that arise from the in-kind subscription appear in the next portfolio snapshot (Chapter 6) and are conventionally annotated through `<TransactionSubtype>in-kind</TransactionSubtype>`.

For **negotiated institutional blocks** — very large trades priced individually between the fund and a counterparty rather than at the standard NAV — the same mechanism applies: an ordinary `<Flow>` carrying the negotiated value, with `<TransactionSubtype>negotiated</TransactionSubtype>` as annotation.

7.9.4 The EMT Regulatory Templates and Cost Disclosure

Chapter 8 treats the FinDatEx regulatory templates that the Europa Growth Fund embeds in the `<RegulatoryReportings>` root element. The **European MiFID Template** (EMT) is the relevant one for cost disclosure: it carries the entry costs, exit costs, ongoing charges, performance fees, and transaction costs that an investor in the share class is subject to, in the structure mandated by MiFID II's cost-disclosure rules.

For producers that need to carry entry-load, exit-load, and swing-pricing data in a **regulatory** context — and that is by far the most common reason for carrying them at all — the EMT fields of Chapter 8 are the authoritative place. For *operational* purposes (telling a distributor what fee to charge on a given subscription), the data lives in the producer's order-management system and is not carried in the FundsXML delivery at all.

The rule for readers of this chapter: if you expect to find entry-load or swing-pricing fields on a `<Flow>` record, you will not find them. The core schema captures the *outcome* of the transaction — the direction, the total value, the dates — and leaves the cost-disclosure mechanics to the regulatory templates.

7.10 The Complete Transaction Block for the Europa Growth Fund

The complete transaction data for the reporting period 1–31 March 2026. Because the schema distributes transactions across three containers at two levels of the fund element, the "complete transaction block" is not one listing — it is **four** separate listings: three `<Flows>` containers (one per share class) and one `<Earnings>` container (at the fund-level portfolio). Structured in four subsections.

7.10.1 Overview of the Period

During March 2026 the Europa Growth Fund saw normal transaction activity for a mid-sized European UCITS: seven subscriptions (a net inflow), four redemptions against them, one intra-fund switch, and three dividend income events from European portfolio holdings that paid in Q1. **No distribution-paid events** — the next one is in mid-May, as §7.7 explained.

The fifteen conceptual events map to sixteen schema records, because the one intra-fund switch is represented as two linked `<Flow>` entries in two different share classes. The full breakdown is:

- **R-EUR-ACC `<Flows>` (six `<Flow>` entries):** 3 subscriptions (03, 16, 23 March), 2 redemptions (06, 13 March), 1 switch source leg (12 March, paired with the I-EUR-DIST target leg).
- **R-CHF-ACC-HEDGED `<Flows>` (three `<Flow>` entries):** 2 subscriptions (05, 26 March), 1 redemption (19 March). No switch involvement.
- **I-EUR-DIST `<Flows>` (four `<Flow>` entries):** 2 subscriptions (09, 30 March), 1 redemption (20 March), 1 switch target leg (12 March, paired with the R-EUR-ACC source leg).

- **Fund-level portfolio <Earnings> (three <Earning> entries):** Q1 dividends from Allianz (11 March, German), BNP Paribas (20 March, French), and AstraZeneca (27 March, British).

The events are grouped by container below, not by chronological order across the whole delivery, because the schema nesting demands it. Within each container, entries are listed by date. The complete listing in its fully-populated form has been validated against `FundsXML4.xsd` as an end-to-end document with a matching `AssetMasterData` block carrying the three dividend-paying assets.

7.10.2 The R-EUR-ACC Flows Block

The six `<Flow>` entries inside the R-EUR-ACC share class's `<Flows>` container. Three subscriptions, two ordinary redemptions, and the source leg of the switch.

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000011] -->
<Flows from="2026-03-01" to="2026-03-31">
  <!-- Subscriptions -->
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260303-0001</TransactionID>
    <TradeDate>2026-03-03</TradeDate>
    <AccountingDate>2026-03-03</AccountingDate>
    <SettlementDate>2026-03-05</SettlementDate>
    <ValueDate>2026-03-03</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>202.1578</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">25000.00</Amount></TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260316-0024</TransactionID>
    <TradeDate>2026-03-16</TradeDate>
    <AccountingDate>2026-03-16</AccountingDate>
    <SettlementDate>2026-03-18</SettlementDate>
    <ValueDate>2026-03-16</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>80.6852</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">10000.00</Amount></TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260323-0032</TransactionID>
    <TradeDate>2026-03-23</TradeDate>
    <AccountingDate>2026-03-23</AccountingDate>
    <SettlementDate>2026-03-25</SettlementDate>
    <ValueDate>2026-03-23</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>402.4811</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">50000.00</Amount></TotalValue>
```



```

    <Channel>Retail</Channel>
  </Flow>
  <!-- Redemptions -->
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260306-0004</TransactionID>
    <TradeDate>2026-03-06</TradeDate>
    <AccountingDate>2026-03-06</AccountingDate>
    <SettlementDate>2026-03-10</SettlementDate>
    <ValueDate>2026-03-06</ValueDate>
    <TransactionType>RED</TransactionType>
    <Units>68.7247</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">8500.00</Amount></TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260313-0017</TransactionID>
    <TradeDate>2026-03-13</TradeDate>
    <AccountingDate>2026-03-13</AccountingDate>
    <SettlementDate>2026-03-17</SettlementDate>
    <ValueDate>2026-03-13</ValueDate>
    <TransactionType>RED</TransactionType>
    <Units>258.4821</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">32000.00</Amount></TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <!-- Switch, source leg (paired with EGF-SW-20260312-0007-TGT in the I-EUR-DIST
class) -->
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-SW-20260312-0007-SRC</TransactionID>
    <TradeDate>2026-03-12</TradeDate>
    <AccountingDate>2026-03-12</AccountingDate>
    <SettlementDate>2026-03-16</SettlementDate>
    <ValueDate>2026-03-12</ValueDate>
    <TransactionType>RED</TransactionType>
    <TransactionSubtype>switch – source leg; paired with EGF-SW-20260312-0007-TGT on
LU2100000037</TransactionSubtype>
    <Units>404.2156</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">50000.00</Amount></TotalValue>
    <Channel>Intra-fund switch</Channel>
  </Flow>
</Flows>

```

7.10.3 The R-CHF-ACC-HEDGED Flows Block

Three `<Flow>` entries inside the R-CHF-ACC-HEDGED share class's `<Flows>` container. Two subscriptions and one redemption, all in Swiss francs. The `<TotalValue>` carries two `<Amount>` children in each entry — one in CHF flagged `isShareClassCcy="true"`, one in EUR flagged `isFundCcy="true"` — so that consumers read-

ing either currency see a consistent figure.

```
<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000029] -->
<Flows from="2026-03-01" to="2026-03-31">
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260305-0003</TransactionID>
    <TradeDate>2026-03-05</TradeDate>
    <AccountingDate>2026-03-05</AccountingDate>
    <SettlementDate>2026-03-09</SettlementDate>
    <ValueDate>2026-03-05</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>125.8812</Units>
    <TotalValue>
      <Amount ccy="CHF" isShareClassCcy="true">15000.00</Amount>
      <Amount ccy="EUR" isFundCcy="true">14410.75</Amount>
    </TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260326-0038</TransactionID>
    <TradeDate>2026-03-26</TradeDate>
    <AccountingDate>2026-03-26</AccountingDate>
    <SettlementDate>2026-03-30</SettlementDate>
    <ValueDate>2026-03-26</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>335.6181</Units>
    <TotalValue>
      <Amount ccy="CHF" isShareClassCcy="true">40000.00</Amount>
      <Amount ccy="EUR" isFundCcy="true">38428.00</Amount>
    </TotalValue>
    <Channel>Retail</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260319-0022</TransactionID>
    <TradeDate>2026-03-19</TradeDate>
    <AccountingDate>2026-03-19</AccountingDate>
    <SettlementDate>2026-03-23</SettlementDate>
    <ValueDate>2026-03-19</ValueDate>
    <TransactionType>RED</TransactionType>
    <Units>180.4108</Units>
    <TotalValue>
      <Amount ccy="CHF" isShareClassCcy="true">21500.00</Amount>
      <Amount ccy="EUR" isFundCcy="true">20654.68</Amount>
    </TotalValue>
    <Channel>Retail</Channel>
  </Flow>
</Flows>
```

7.10.4 The I-EUR-DIST Flows Block

Four `<Flow>` entries inside the I-EUR-DIST share class's `<Flows>` container. Two institutional subscriptions, one institutional redemption, and the target leg of the switch.

```

<!-- Inside Fund/SingleFund/ShareClasses/ShareClass[ISIN=LU2100000037] -->
<Flows from="2026-03-01" to="2026-03-31">
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260309-0005</TransactionID>
    <TradeDate>2026-03-09</TradeDate>
    <AccountingDate>2026-03-09</AccountingDate>
    <SettlementDate>2026-03-11</SettlementDate>
    <ValueDate>2026-03-09</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>23202.5830</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">2500000.00</Amount></TotalValue>
    <Channel>Institutional</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260330-0043</TransactionID>
    <TradeDate>2026-03-30</TradeDate>
    <AccountingDate>2026-03-30</AccountingDate>
    <SettlementDate>2026-04-01</SettlementDate>
    <ValueDate>2026-03-30</ValueDate>
    <TransactionType>SUB</TransactionType>
    <Units>46151.2845</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">5000000.00</Amount></TotalValue>
    <Channel>Institutional</Channel>
  </Flow>
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-TX-20260320-0011</TransactionID>
    <TradeDate>2026-03-20</TradeDate>
    <AccountingDate>2026-03-20</AccountingDate>
    <SettlementDate>2026-03-24</SettlementDate>
    <ValueDate>2026-03-20</ValueDate>
    <TransactionType>RED</TransactionType>
    <Units>15000.0000</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">1617631.50</Amount></TotalValue>
    <Channel>Institutional</Channel>
  </Flow>
  <!-- Switch, target leg (paired with EGF-SW-20260312-0007-SRC in the R-EUR-ACC class)
  -->
  <Flow>
    <ActionCode>C</ActionCode>
    <TransactionID>EGF-SW-20260312-0007-TGT</TransactionID>
    <TradeDate>2026-03-12</TradeDate>
    <AccountingDate>2026-03-12</AccountingDate>
    <SettlementDate>2026-03-16</SettlementDate>

```

```

    <ValueDate>2026-03-12</ValueDate>
    <TransactionType>SUB</TransactionType>
    <TransactionSubtype>switch – target leg; paired with EGF-SW-20260312-0007-SRC on
LU2100000011</TransactionSubtype>
    <Units>464.0882</Units>
    <TotalValue><Amount ccy="EUR" isFundCcy="true">50000.00</Amount></TotalValue>
    <Channel>Intra-fund switch</Channel>
  </Flow>
</Flows>

```

7.10.5 The Fund-Level Portfolio Earnings Block

Three `<Earning>` entries inside the fund-level `<Portfolio>/<Earnings>` container. These are the Q1 2026 dividend receipts from three European portfolio holdings, distinguished by source country through the different withholding-tax regimes. The German entry uses `WithholdingTaxQuota=26.375` (Germany's standard withholding rate), the French entry uses `25.00` (the France-Luxembourg tax-treaty rate), and the British entry uses `0.00` (the United Kingdom does not withhold tax on dividends paid to non-resident shareholders).

```

<!-- Inside Fund/FundDynamicData/Portfolios/Portfolio -->
<Earnings from="2026-03-01" to="2026-03-31">
  <Earning>
    <EarningID>EGF-EARN-20260311-0008</EarningID>
    <AssetUniqueID>DE0008404005</AssetUniqueID>
    <EarningKind>Dividend</EarningKind>
    <ClosingDate>2026-03-11</ClosingDate>
    <EntryDate>2026-03-13</EntryDate>
    <ValutaDate>2026-03-13</ValutaDate>
    <EntryCurrency>EUR</EntryCurrency>
    <IncomeCurrency>EUR</IncomeCurrency>
    <EntryValue>
      <Amount ccy="EUR" isFundCcy="true">362830.00</Amount>
    </EntryValue>
    <NominalValueOrUnits>32000</NominalValueOrUnits>
    <PostingText>Allianz SE 2026 annual dividend, net of 26.375% German withholding
tax</PostingText>
    <Dividend>
      <DividendPerUnitValue><Amount ccy="EUR">15.40</Amount></DividendPerUnitValue>
      <DividendGrossValue><Amount ccy="EUR" isFundCcy="true">492800.00</Amount>
    </DividendGrossValue>
      <DividendNetValue><Amount ccy="EUR" isFundCcy="true">362830.00</Amount>
    </DividendNetValue>
      <WithholdingTaxValue><Amount ccy="EUR">129970.00</Amount></WithholdingTaxValue>
      <ExDay>2026-03-11</ExDay>
      <PayDay>2026-03-13</PayDay>
      <WithholdingTaxQuota>26.375</WithholdingTaxQuota>
    </Dividend>
  </Earning>
  <Earning>
    <EarningID>EGF-EARN-20260320-0012</EarningID>

```

```

<AssetUniqueID>FR0000131104</AssetUniqueID>
<EarningKind>Dividend</EarningKind>
<ClosingDate>2026-03-20</ClosingDate>
<EntryDate>2026-03-24</EntryDate>
<ValutaDate>2026-03-24</ValutaDate>
<EntryCurrency>EUR</EntryCurrency>
<IncomeCurrency>EUR</IncomeCurrency>
<EntryValue>
  <Amount ccy="EUR" isFundCcy="true">123750.00</Amount>
</EntryValue>
<NominalValueOrUnits>110000</NominalValueOrUnits>
<PostingText>BNP Paribas Q1 interim dividend, net of 25% French withholding
tax</PostingText>
<Dividend>
  <DividendPerUnitValue><Amount ccy="EUR">1.50</Amount></DividendPerUnitValue>
  <DividendGrossValue><Amount ccy="EUR" isFundCcy="true">165000.00</Amount>
</DividendGrossValue>
  <DividendNetValue><Amount ccy="EUR" isFundCcy="true">123750.00</Amount>
</DividendNetValue>
  <WithholdingTaxValue><Amount ccy="EUR">41250.00</Amount></WithholdingTaxValue>
  <ExDay>2026-03-20</ExDay>
  <PayDay>2026-03-24</PayDay>
  <WithholdingTaxQuota>25.00</WithholdingTaxQuota>
</Dividend>
</Earning>
<Earning>
  <EarningID>EGF-EARN-20260327-0015</EarningID>
  <AssetUniqueID>GB0009895292</AssetUniqueID>
  <EarningKind>Dividend</EarningKind>
  <ClosingDate>2026-03-27</ClosingDate>
  <EntryDate>2026-03-31</EntryDate>
  <ValutaDate>2026-03-31</ValutaDate>
  <EntryCurrency>GBP</EntryCurrency>
  <IncomeCurrency>GBP</IncomeCurrency>
  <EntryValue>
    <Amount ccy="GBP">69000.00</Amount>
    <Amount ccy="EUR" isFundCcy="true">81742.50</Amount>
  </EntryValue>
  <NominalValueOrUnits>75000</NominalValueOrUnits>
  <PostingText>AstraZeneca interim dividend, no UK withholding tax on non-resident
dividends</PostingText>
  <Dividend>
    <DividendPerUnitValue><Amount ccy="GBP">0.92</Amount></DividendPerUnitValue>
    <DividendGrossValue><Amount ccy="GBP">69000.00</Amount></DividendGrossValue>
    <DividendNetValue><Amount ccy="GBP">69000.00</Amount></DividendNetValue>
    <WithholdingTaxValue><Amount ccy="GBP">0.00</Amount></WithholdingTaxValue>
    <ExDay>2026-03-27</ExDay>
    <PayDay>2026-03-31</PayDay>
    <WithholdingTaxQuota>0.00</WithholdingTaxQuota>
  </Dividend>

```

```
</Earning>
</Earnings>
```

7.10.6 A Three-Pass Reading

The four blocks are best understood in three passes, one per conceptual group.

First pass — investor-side flows. Eleven `<Flow>` entries representing seven subscriptions and four redemptions, distributed across the three share classes' `<Flows>` containers. The net flow is positive: subscriptions total roughly 7.64 million EUR equivalent (counting the CHF figures via the embedded EUR Amounts in `<TotalValue>`), while redemptions total roughly 1.67 million EUR equivalent. The net inflow is therefore around 6 million EUR for the month. This explains why the `SharesOutstanding` of all three classes grew between 28 February (roughly 4.01 million shares total) and 31 March (the 1,234,567 + 456,789 + 2,345,678 figures from Chapter 5). The institutional class I-EUR-DIST saw the largest single inflows (a 2.5 million EUR subscription on 9 March and a 5 million EUR subscription on 30 March), consistent with the notion that institutional investors make larger but less frequent allocations than retail investors. None of the redemptions triggered swing pricing, because none of them approached the 2% net-outflow threshold — but as §7.5.2 noted, swing-pricing data does not appear in the schema-level `<Flow>` records anyway, so a consumer that wants to verify the no-swing condition reads a `<CustomAttributes>` field on the share class instead.

Second pass — the switch as two linked Flows. One intra-fund switch represented as two `<Flow>` entries with a `-SRC / -TGT` `<TransactionID>` pairing convention. The source leg sits in the R-EUR-ACC class's `<Flows>` (the fifth entry of §7.10.2's listing) with `<TransactionType>RED</TransactionType>` and 404.2156 units; the target leg sits in the I-EUR-DIST class's `<Flows>` (the fourth entry of §7.10.4's listing) with `<TransactionType>SUB</TransactionType>` and 464.0882 units. Both legs carry the same `<TotalValue>` of 50,000 EUR. The two `<Units>` values differ because the two classes have different NAVs, and both are positive — direction comes from `<TransactionType>`. The pairing convention lives in `<TransactionSubtype>`, which on each leg names the paired leg's ID. A consumer that walks the document linearly sees the two legs as a redemption and a subscription that happen to net to zero at the fund level; a consumer that recognises the pairing reconstructs the conceptual switch. It is technically unlikely that a 50,000 EUR stake would be allowed to cross into the institutional class (the minimum is 1,000,000 EUR), but the example is retained for structural illustration.

Third pass — income events in the portfolio. Three `<Earning>` entries inside the fund-level `<Portfolio>/<Earnings>` container, each representing a Q1 dividend from a different country. Each carries its own withholding-tax regime: 26.375% for Germany (Allianz), 25% for France (BNP Paribas), and 0% for the United Kingdom (AstraZeneca; the United Kingdom does not withhold tax on dividends paid to non-resident shareholders, a peculiarity of UK dividend taxation). The gross amounts total 492,800 EUR + 165,000 EUR + 69,000 GBP ≈ 740,000 EUR equivalent; the net amounts total around 570,000 EUR after withholding. These amounts increase the fund's cash balance and contribute to the Q1 income that will eventually feed the May distribution on the I-EUR-DIST class. Each `<Earning>` carries an `<AssetUniqueID>` IDREF pointing at the corresponding entry in `<AssetMasterData>` (Chapter 6), so that a consumer can join the dividend back to the position that earned it without ambiguity.

The four blocks as shown reconcile internally: the sum of `<TotalValue>` deltas across the eleven `<Flow>` entries and the cash inflows from the three `<Earning>` entries are consistent with the movements in `<TotalAssetValues>` between the two valuation points. The complete reconciliation — starting from the 28

February state, applying every flow and every earning in these blocks, arriving at the 31 March state — is printed in Appendix D as a worked example, using the validated end-to-end document this chapter is built from.

7.11 Common Pitfalls

The following short list captures the mistakes that, in our experience, cause the greatest share of transaction-related production incidents.

- **Looking for transactions in the wrong container.** A consumer expects investor flows under `FundDynamicData` and finds nothing, because the real container is `Fund/SingleFund/ShareClasses/ShareClass/Flows`. Or expects portfolio income events under a share class and finds nothing, because the real container is `Fund/FundDynamicData/Portfolios/Portfolio/Earnings`. The rule: `<Flows>` and `<Distributions>` live on each share class; `<Earnings>` lives on the fund-level portfolio; there is no fund-level `<SingleFundFlows>`-style aggregator.
- **Negative `<Units>` on a redemption.** A producer treats `<Units>` as a signed number to indicate direction. The schema explicitly demands positive values: "Number of shares/units bought or sold (should be always positive)." Direction comes from `<TransactionType>` (`SUB` or `RED`), not from the sign of `<Units>`. A consumer that sums signed units mis-handles the producer's positive-only data and gets the wrong inflow figure.
- **Missing `<ActionCode>`, `<AccountingDate>`, `<TransactionType>`, or `<TotalValue>` on a `<Flow>`.** All four are mandatory in `FlowType`. A producer that omits any of them fails schema validation at load time. The most common omission is `<AccountingDate>`, because legacy systems carry only a "trade date" concept and forget to map it to the schema's accounting-date field.
- **Switch counted as one transaction in either direction.** A consumer sees only the source leg (`<TransactionType>RED</TransactionType>`) of a switch and reports a redemption to the producer's records, without realising that the matching target leg lives in another share class's `<Flows>`. The net flow metric is wrongly negative. The rule: a switch is **two** linked `<Flow>` entries, and a consumer that wants to detect them looks for the `<TransactionSubtype>` annotation or for paired `<TransactionID>` prefixes across share classes.
- **`AnnouncementDate` written as `DeclarationDate`.** A producer migrating from a legacy "declaration date" vocabulary writes `<DeclarationDate>` on a `<Distribution>` and fails schema validation. The schema element is `<AnnouncementDate>`. Similar trap: the schema uses `ExDate`, `RecordDate`, `PaymentDate` as element names — the operational concepts have schema names that do not always match the textbook vocabulary.
- **`<DividendStatus>` missing on a distribution.** Mandatory enum (`ESTIMATED` or `OFFICIAL`). A producer that forgets it fails validation. A consumer that finds it set to `ESTIMATED` should expect the same distribution to reappear in a later delivery with `OFFICIAL` status; treating an `ESTIMATED` distribution as the final value is a recurring source of investor-statement errors.
- **`PerShare` written without the `Total` it pairs with.** `GrossDividendAmount` and `NetDividendAmount` each have an optional `<Total>` and a mandatory `<PerShare>`. The structure is gross: `<GrossDividendAmount><Total>...</Total><PerShare>...</PerShare>`

`</GrossDividendAmount>` . A producer that emits

`<GrossDividendAmount>3.80</GrossDividendAmount>` flat (without the inner `<PerShare>` wrapper) fails validation.

- **Distributions confused with earnings.** A producer or consumer treats a dividend received from a portfolio holding (an `<Earning>` in the portfolio) as if it were a distribution paid to investors (a `<Distribution>` in a share class), or vice versa. The two are economic opposites and live in different containers. The rule: cash flows *into* the fund go into `Portfolio/Earnings` ; cash flows *out of* a share class to its investors go into `ShareClass/Distributions` .
- **Withholding tax forgotten on income events.** A consumer sums the `<DividendGrossValue>` fields of `<Earning>` records without subtracting the `<WithholdingTaxValue>` and computes the fund's cash balance too high. Reconciliation with the custodian statement fails. The rule: use `<EntryValue>` (or `<DividendNetValue>`) for cash reconciliation, `<DividendGrossValue>` for income analysis.
- **`<EarningKind>` value misspelled.** The schema enumeration is `Coupon` , `Dividend` , `Fund distribution` , `Interest deposit/giro` , `Interest swap` , `Other` . Producers that write `Fund Distribution` (capital D), `InterestDeposit` , or `Interests` fail validation. The exact spelling — including the embedded slash and the lowercase second word — is part of the schema.
- **`<SettlementDate>` values treated as trade dates.** A consumer archives the `<SettlementDate>` of a `<Flow>` as the trade date, and later queries for "which orders came in on day X?" return wrong results. The rule: `<TradeDate>` for audit and reporting, `<AccountingDate>` for NAV pricing, `<SettlementDate>` only for cash movements.
- **`OrderExecutionType` written as a `<Flow>` child.** A producer migrating from another fund-data format writes `<OrderExecutionType>AtNAV</OrderExecutionType>` and fails schema validation, because no such element exists. The operational concept (forward pricing, cut-offs, in-kind subscription, exchange execution) is not modelled in the core schema; producers that need to carry it use `<CustomAttributes>` or the regulatory templates of Chapter 8.

7.12 Key Takeaways

- Transactions are not centralised in FundsXML. They live in three different containers attached to two different parents: subscriptions and redemptions in `ShareClass/Flows/Flow` (schema type `FlowType`), distributions paid to investors in `ShareClass/Distributions/Distribution` (`DistributionType`), and income received from portfolio holdings in `Portfolio/Earnings/Earning` (`EarningType`). There is no fund-level aggregator container.
- `FlowType` carries four mandatory children — `<ActionCode>` , `<AccountingDate>` , `<TransactionType>` , `<TotalValue>` — and a binary `<TransactionType>` enumeration with only two values: `SUB` and `RED` . The schema does not define a third value for switches, distributions, or income; those event types either live in other containers or are modelled as paired `SUB / RED` records.
- Every `<Flow>` carries up to four date fields: `<TradeDate>` , `<AccountingDate>` , `<SettlementDate>` , `<ValueDate>` . Only `<AccountingDate>` is mandatory. There is no element called `EffectiveDate` ; the legacy textbook *effective date* maps to `<AccountingDate>` , the field whose value carries the cut-off decision in the European forward-pricing model.

- `<Units>` on `<Flow>` is always positive, by explicit schema convention. Direction comes from `<TransactionType>`. `<TotalValue>` is the one mandatory money amount and uses `FundAmountType`, so it can carry several `<Amount>` children at once — typically one in the fund's base currency and one in the share-class currency for hedged or non-base-currency classes.
- Switches are not a first-class schema construct. The schema has no `Switch` value in `<TransactionType>`; an intra-fund switch is modelled as **two linked `<Flow>` entries**, one with `RED` on the source class and one with `SUB` on the target class. The convention for linking the two is shared `<TransactionID>` prefix and matching `<TransactionSubtype>` annotations. Inter-fund switches use the same pattern across two different `<Fund>` elements.
- `DistributionType` (in `ShareClass/Distributions`) carries seven mandatory children: `<ActionCode>`, `<DividendStatus>` (`ESTIMATED / OFFICIAL`), `<ExDate>`, `<PaymentDate>`, `<PaymentCurrency>`, `<GrossDividendAmount>`, `<NetDividendAmount>`. The two amount blocks each contain an optional `<Total>` and a mandatory `<PerShare>`. The schema element is `<AnnouncementDate>`, not the legacy `DeclarationDate`. There is no `DistributionSubType` enumeration.
- `EarningType` (in `Portfolio/Earnings`) carries five mandatory children: `<EarningID>`, `<EarningKind>`, `<EntryDate>`, `<EntryCurrency>`, `<EntryValue>`. `<EarningKind>` is a six-value enumeration: `Coupon`, `Dividend`, `Fund distribution`, `Interest deposit/giro`, `Interest swap`, `Other`. Each of the first three has a typed sub-block (`<Coupon>`, `<Dividend>`, `<FundDistribution>`) carrying the kind-specific breakdown including withholding-tax fields.
- The `<AssetUniqueID>` field on an `<Earning>` is an `xs:IDREF` pointing at the matching `<Asset>` in `<AssetMasterData>` — the same parser-enforced linkage mechanism as `<Position>/<UniqueID>` from Chapter 6.
- The forward-pricing concepts (cut-off times, swing pricing, entry/exit loads, in-kind creation, exchange execution) are real and operationally important, but the FundsXML 4.2.8 core schema does not model them on `<Flow>`. Producers that need to carry them use `<CustomAttributes>` on the share class or the EMT regulatory templates of Chapter 8.
- The Europa Growth Fund's March 2026 transaction block consists of three `<Flows>` containers (six entries on R-EUR-ACC, three on R-CHF-ACC-HEDGED, four on I-EUR-DIST — sixteen Flow entries in total covering eleven trades and two switch legs) plus three `<Earning>` entries on the fund-level portfolio. There are no distributions, because the next one does not occur until mid-May.

We have now covered the three transaction containers — `ShareClass/Flows`, `ShareClass/Distributions`, and `Portfolio/Earnings` — together with the per-instrument trade container `Portfolio/Transactions` that we deferred to Chapter 12. What we have not yet covered is the fifth main area of a FundsXML delivery: `RegulatoryReportings`, where the five FinDatEx templates live. Chapter 8 opens that area and walks through EMT, EPT, EET, EFT, and TPT — the regulatory disclosures that modern European fund distribution depends on, and the place where many of the operational concepts of §7.9 actually find their FundsXML home.



Regulatory Modules

EMT, EPT, EET, EFT, and TPT in FundsXML

8.1 Setting the Scene: The Fifth Area

At the end of Chapter 7 we had covered four of the five main areas of a FundsXML delivery: `ControlData`, `Funds`, `AssetMasterData`, and the substructures inside `FundDynamicData`. The fifth main area — `RegulatoryReportings` — has been waiting since Chapter 3, mentioned in passing but never treated in its own right. This chapter fills that gap, and it fills it at length, because the regulatory modules that live inside `RegulatoryReportings` are both varied in their subject matter and unavoidable in their practical importance. A European fund that cannot deliver its EMT cannot be retailed under MiFID II. A fund that cannot deliver its EPT cannot publish a PRIIPs KID. A fund that cannot deliver its EET cannot qualify for SFDR-classified distribution. A fund that cannot deliver its TPT cannot be meaningfully held by insurance investors under Solvency II. The templates behind these acronyms are, collectively, the regulatory operating system of the European fund industry.

The chapter covers all five FinDatEx templates — EMT, EPT, EET, EFT, and TPT — together with the European Single Access Point (ESAP) that is in the process of becoming the EU-wide gateway for regulatory disclosures. The treatment is weighted by practical importance: EMT and EET, the two most operationally consequential templates, receive the most attention; EPT follows closely; TPT and EFT are lighter; ESAP rounds the chapter off with a short section.

Before we begin, one honest framing deserves to be stated up front. This is not a chapter about PRIIPs, SFDR, or MiFID II *as regulations*. Entire books are written about each of those topics, and readers who need the full regulatory depth should consult them; Appendix E lists the primary sources. This chapter treats the **FundsXML embedding** of the FinDatEx templates — the XML structure, the main field groups, the relationship to the rest of a FundsXML document, and enough regulatory context (a page or two per template) that the reader can populate the fields correctly for a realistic fund. For the deep regulatory interpretation of any particular field, the authoritative source is always the current FinDatEx template specification and the corresponding ESMA technical standards.

By the end of this chapter, you should be able to:

- explain how `RegulatoryReportings` sits in a FundsXML document and how it splits into `DirectReporting` and `IndirectReporting` branches;
- name the regulatory driver of each template (MiFID II for EMT, PRIIPs for EPT, SFDR for EET, product governance for EFT, Solvency II for TPT);
- populate the main fields of each template for a realistic European UCITS;
- distinguish the cost fields of EMT from the cost fields of EPT and know which consumer uses which;
- classify a fund as SFDR Article 6, 8, or 9 and populate the corresponding EET fields;
- explain what ESAP is and how FundsXML deliveries may reach it;

- read a complete regulatory-reporting block and recognise which templates are present, which are absent, and why.

The Europa Growth Fund continues as the running example through all five templates. The fund is natively well-suited to the regulatory agenda of this chapter: it is a UCITS sold across eleven distribution countries (EMT and EPT), it carries an SFDR Article 8 classification (EET), it is held by European insurance investors (TPT), and it receives regular feedback from its distributors (EFT, though EFT will not appear in the March 2026 delivery for reasons explained in §8.6).

8.2 RegulatoryReportings and the FinDatEx Consortium

Before we look at any individual template, the structural question: where, inside a FundsXML document, does `RegulatoryReportings` sit, and who decides what goes into it?

8.2.1 The Structural Location

`<RegulatoryReportings>` is a root-level element of a FundsXML document, a sibling of `<Funds>` and `<AssetMasterData>`. It does *not* live inside `<FundDynamicData>` alongside `TotalAssetValues`, `Portfolios`, and `SingleFundFlows`. The choice of position is deliberate: regulatory modules have their own release cadence, their own semantics, and their own consumer audiences, and coupling them tightly to the valuation-point-driven dynamic data would be the wrong design. A fund may emit a fresh EMT every month without changing its NAV, and a fund may publish a new daily NAV without changing its EMT. The two are decoupled, and the schema expresses the decoupling by placing them in separate branches of the document.

Inside `<RegulatoryReportings>` the schema immediately splits into two parallel branches — `DirectReporting` and `IndirectReporting` — and the five FinDatEx templates are distributed between them:

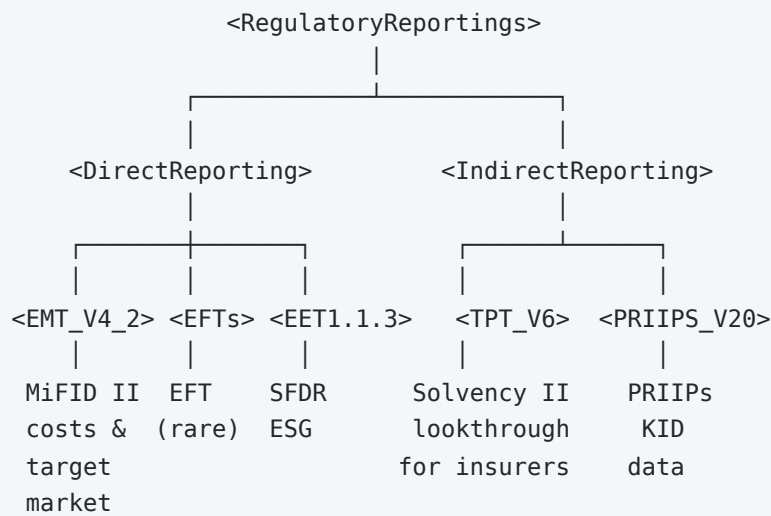
```
<FundsXML4>
  <ControlData>...</ControlData>           <!-- Chapter 4 -->
  <Funds>
    <Fund>
      <FundStaticData>...</FundStaticData>   <!-- Chapter 5 -->
      <FundDynamicData>
        <TotalAssetValues>...</TotalAssetValues> <!-- Chapter 5 -->
        <Portfolios>...</Portfolios>         <!-- Chapter 6 -->
        <SingleFundFlows>...</SingleFundFlows> <!-- Chapter 7 -->
      </FundDynamicData>
    </Fund>
  </Funds>
  <AssetMasterData>...</AssetMasterData>     <!-- Chapter 6 -->
  <RegulatoryReportings>                     <!-- this chapter -->
    <DirectReporting>
      <EMT_V4_2>...</EMT_V4_2>              <!-- MiFID II -->
      <EFTs>...</EFTs>                     <!-- product governance -->
```

```

    <EET1.1.3>...</EET1.1.3>                                <!-- SFDR -->
  </DirectReporting>
  <IndirectReporting>
    <TripartiteTemplateSolvencyII_V6>...
    </TripartiteTemplateSolvencyII_V6>                    <!-- Solvency II -->
    <PRIIPS_V20>...</PRIIPS_V20>                        <!-- PRIIPs KID -->
  </IndirectReporting>
</RegulatoryReportings>
</FundsXML4>

```

Figure 8.1 — The RegulatoryReportings block



The split between `DirectReporting` and `IndirectReporting` reflects the intended distribution path of each template. `DirectReporting` contains the templates that flow directly between regulated counterparties — EMT from manufacturer to distributor, EFT from distributor to manufacturer, EET as an SFDR disclosure artefact shared bilaterally. `IndirectReporting` contains the templates that flow through an intermediated disclosure chain — PRIIPs data, which ultimately reaches retail investors via a KID generator or an aggregator; TPT, which reaches insurance investors via their Solvency II reporting pipeline. Both branches are optional, and a fund that emits only one branch (or neither) is still schema-valid.

The critical rule is that **each module is self-contained**. EMT does not depend on EPT; EET does not know about EFT; TPT is independent of all the others. A consumer that processes only EMT reads only the `<EMT_V4_2>` branch; a consumer that needs TPT for Solvency II purposes reads only the `<TripartiteTemplateSolvencyII_V6>` branch. The modules share the enclosing `<RegulatoryReportings>` container and nothing else. There is no cross-module linking, no shared lookup tables, no implicit dependencies between the five blocks.

A corollary of this independence is that a `<RegulatoryReportings>` block may contain any subset of the five modules. A fund that has no insurance investors omits TPT. A fund that receives no distributor feedback in the reporting period omits EFT. A fund that is not actively subject to SFDR disclosures (a rare case, but it exists for some AIFs) omits EET. The schema tolerates any combination of the five, and consumers are expected to cope with partial blocks.

A brief note on the versioned element names that will appear throughout this chapter. `EMT_V4_2` corresponds to FinDatEx European MiFID Template version 4.2 of April 2024. `PRIIPS_V20` is the FinDatEx European PRIIPs Template v2.0. `EET1.1.3` is the European ESG Template v1.1.3. `EFTs` is the container for the European Feedback Template v1.0. `TripartiteTemplateSolvencyII_V6` is the Tripartite Template v6.0 of March 2022. Each versioned element has its own type definition in the FundsXML XSD, and the schema retains every historical version (`EMT` , `EMT_V3` , `EMT_V4` , `EMT_V4_1` , `EMT_V4_2` ; `EET1.0` , `EET1.1` , `EET1.1.1` , `EET1.1.2` , `EET1.1.3` ; and so on) so that older documents continue to validate. A producer targeting a contemporary consumer uses the latest version on each line.

8.2.2 The FinDatEx Consortium

A point of great importance that deserves to be stated plainly: **FinDatEx is not FundsXML**. FinDatEx is an independent industry body, formed in 2018 and 2019 by the principal European financial-services associations — EFAMA for the fund industry, EBF for banks, Insurance Europe for insurers, among others — with the single mission of defining and maintaining standardised data templates to meet the regulatory disclosure requirements of the European financial sector. The templates are published and governed by FinDatEx. FundsXML embeds them.

The division of labour is worth spelling out because it determines who is responsible for what:

- **FinDatEx** decides the *content* of the templates: which fields exist, what they mean, what enumerated values they take, which validation rules apply at the semantic level. When a new PRIIPs technical standard is adopted by ESMA, FinDatEx updates the EPT to reflect the new field set. When SFDR Level 2 RTS introduces a new mandatory PAI indicator, FinDatEx updates the EET. The changes are published as new template versions — typically annually, with hotfix releases when regulatory changes demand them — in the form of Excel specifications that list every field, its type, and its regulatory meaning.
- **FundsXML** decides the *form* in which the templates are embedded: the XML element structure, the complex types, the way enumerations are serialised, and how each new FinDatEx version is added as a new, side-by-side versioned element under `DirectReporting` or `IndirectReporting`. When FinDatEx publishes a new EMT version, FundsXML follows in its next minor release, incorporating the new fields into its XSD and exposing the new version as a sibling element next to the previous one.

The separation is clean and the two organisations work closely together. Chapter 3 introduced this relationship in §3.2.3 as one of the patterns of FundsXML's history — regulation drives a new payload, FundsXML absorbs the payload as a new module. This chapter is where the pattern becomes operational: the five templates that follow are each an example of a regulation-driven module absorbed into a stable envelope.

An important practical consequence of the separation: **disputes about field meaning are settled at FinDatEx, not at FundsXML**. A producer who is unsure how a particular EMT cost field should be populated for a complex fee structure should consult the current FinDatEx EMT specification and its field-level guidance, not the FundsXML schema documentation. The FundsXML schema tells the producer *where* to put the value; the FinDatEx specification tells the producer *what* value to put there.

8.2.3 Module Selection — What to Send When

Not every FundsXML delivery carries all five templates. Which templates are included depends on the purpose of the delivery and on the needs of its consumers. The conventions that govern module selection are stable enough to be summarised.

EMT is the most frequently updated of the five. A fund that is actively retailed across the European Union typically emits a fresh EMT in every monthly delivery, and more frequently when something in its cost structure or target market changes (a new share class is launched, a fee is revised, a distribution country is added). A consumer that runs a retail distribution platform expects to find EMT in every delivery from every manufacturer it has a contract with.

EPT follows the PRIIPs KID refresh cycle. The KID itself must be reviewed at least annually, and re-issued whenever anything material changes (a significant shift in SRI, a new fee structure, a change in recommended holding period). EPT is re-emitted at the same cadence — typically monthly, to keep the downstream KID-generation pipelines synchronised, but the material updates are yearly.

EET runs on a slower cycle than EMT and EPT. SFDR pre-contractual disclosures are updated as the regulatory situation evolves, and PAI values are published on a quarterly or semi-annual basis. A monthly FundsXML delivery usually carries an EET that has been refreshed at most once in the preceding three months; a reporting cycle of quarterly EET updates is common.

EFT, as §8.6 will explain, flows in the opposite direction — from distributor to manufacturer — and therefore appears almost never in a manufacturer-to-distributor delivery. A fund's own monthly delivery does not contain EFT, because the fund is not the one providing feedback to itself. EFT appears in deliveries *from* a distributor *to* the fund's manufacturer, and those are separate files.

TPT is driven by Solvency II reporting dates, which are quarterly and year-end-anchored. An insurance consumer that produces its Solvency II QRT (Quantitative Reporting Template) in the weeks following each quarter-end needs the TPT for the funds it holds. The Europa Growth Fund emits TPT in every monthly delivery, because some of its institutional investors — insurance companies holding the I-EUR-DIST class — need month-end data.

The Europa Growth Fund's delivery of 31 March 2026 contains **EMT, EPT, EET, and TPT** — the four templates that the Europa Asset Management team publishes each month. It does not contain EFT, because no distributor feedback arrived in March that needed to be forwarded. The complete block in §8.9 reflects this pattern.

8.3 EMT — European MiFID Template

The most operationally important of the five templates, and the one that first brought FinDatEx into the foreground of European fund-data exchange.

8.3.1 What EMT Is and Why It Exists

MiFID II — the second Markets in Financial Instruments Directive, in force across the European Union since January 2018 — fundamentally restructured European retail financial advice. Its central theme, repeated in every corner of the legislation, is **transparency**: retail investors should know what they are buying, what it costs, and

whether it is appropriate for them. The regulation places concrete obligations on both sides of the distribution chain. Manufacturers (typically asset managers) must disclose their products in structured detail, and distributors (banks, independent financial advisers, insurance brokers) must pass the disclosures on to investors before a sale and must verify that the investor falls within the manufacturer's intended target market.

The practical consequence is a continuous stream of structured data flowing from every manufacturer to every distributor, for every product, in every jurisdiction. Before the EMT existed, this stream ran over bilateral Excel files in every conceivable format — the exact problem that Chapter 1 cited as the motivation for a standard. A German distributor expected one spreadsheet layout, a French distributor expected another, and a Luxembourg fund administrator maintained dozens of variant templates to keep every bilateral relationship functional. The situation was unsustainable, and FinDatEx published the EMT in 2018 as the common answer. Within two years the European distribution chain had adopted it almost universally, and EMT has since become the *de facto* standard for manufacturer-to-distributor data exchange under MiFID II.

What EMT covers:

- **Product identification** — ISINs, names, currency, manufacturer LEI. Much of this content is repeated from the `FundStaticData` block (Chapter 5), with minor formatting differences to match FinDatEx conventions.
- **Target market** — the five MiFID II dimensions (investor type, knowledge and experience, ability to bear losses, risk tolerance, client objectives). Each dimension is expressed through yes/no/neutral flags so that a distributor can determine which clients the product is intended for and which it is explicitly *not* intended for.
- **Costs and charges** — the full cost structure in the MiFID II format, with one-off costs, ongoing costs, transaction costs, and incidental costs each broken down into their constituent parts. The section splits into *ex ante* (forward-looking) and *ex post* (realised) blocks.
- **Product governance** — the distribution strategy (execution only, with check, investment advice, portfolio management), the recommended holding period, and other governance flags.

EMT is emitted at the **share-class level**, not at the fund level. The Europa Growth Fund, with its three share classes, emits three separate `<FinancialInstrument>` blocks inside the enclosing `<EMT_V4_2>` — one for R-EUR-ACC, one for R-CHF-ACC-HEDGED, and one for I-EUR-DIST — because the cost structure, the target market, and the distribution strategy differ between the classes. The retail classes are aimed at retail investors across eleven countries; the institutional class is aimed at professional investors with a 1,000,000 EUR minimum investment. The same fund, three different disclosures.

8.3.2 Structure of the EMT Module

In the FundsXML embedding, the EMT module is rooted at a `<EMT_V4_2>` element inside `<DirectReporting>`. Its content is a sequence of one or more `<FinancialInstrument>` children, each of which is of type `EMT_V42_FinancialInstrumentType`. The type is structurally rich — nested, not flat — and its sequence has five mandatory top-level sections that together hold the full FinDatEx EMT v4.2 payload:

1. `<DataSetInformation>` — metadata about the EMT delivery itself (version, producer, generation timestamp, and three `Data_Reporting_*` flags that declare which sections the delivery populates);
2. `<GeneralInformation>` — product identification and manufacturer information (ISIN, instrument name, currency, manufacturer name and LEI, product type, and the `Fund` sub-block for UCITS-specific fields);
3. `<TargetMarket>` — the five MiFID II target market dimensions;

4. `<CostsAndChargesExAnte>` — the forward-looking cost disclosure;
5. `<CostsAndChargesExPost>` — the realised cost disclosure (optional, populated only when the `DataReportingExPost` flag in `DataSetInformation` is set to `Y`).

A minimal skeleton of one `<FinancialInstrument>` block looks like this:

```
<EMT_V4_2>
  <FinancialInstrument>
    <DataSetInformation>
      <Version>V4</Version>
      <ProducerName>Europa Asset Management S.A.</ProducerName>
      <ProducerLEI>529900T8BM49AURSD055</ProducerLEI>
      <ProducerEmail>emt@europa-am.lu</ProducerEmail>
      <FileGenerationDateTime>2026-03-31T18:00:00Z</FileGenerationDateTime>
      <DataReportingTargetMarket>Y</DataReportingTargetMarket>
      <DataReportingExAnte>Y</DataReportingExAnte>
      <DataReportingExPost>N</DataReportingExPost>
    </DataSetInformation>
    <GeneralInformation>
      <!-- product identification – see §8.3.3 -->
    </GeneralInformation>
    <TargetMarket>
      <!-- five MiFID II dimensions – see §8.3.4 -->
    </TargetMarket>
    <CostsAndChargesExAnte>
      <!-- cost structure – see §8.3.5 -->
    </CostsAndChargesExAnte>
    <!-- CostsAndChargesExPost is optional -->
  </FinancialInstrument>
</EMT_V4_2>
```

A few structural points are worth noting. The `<Version>` element inside `<DataSetInformation>` uses the FinDatEx internal version string (`V4` , `V4S1` , `V4S2` for EMT version 4 with semantic sub-variants), not the FundsXML element name `EMT_V4_2` . The two are related but serve different purposes: the element name identifies the XSD schema used, and the `<Version>` string identifies the FinDatEx specification being implemented. The three `DataReporting*` flags are a producer's declaration of completeness — they tell the consumer which of the three data sections (target market, ex-ante costs, ex-post costs) have been populated in this particular delivery.

8.3.3 General Information Fields

The `<GeneralInformation>` section carries the identification fields and the manufacturer information. Most of the mandatory content is a rearrangement of what the fund already has in `<FundStaticData>` (Chapter 5), but with FinDatEx-specific formatting conventions that are worth calling out.

```
<GeneralInformation>
  <Code>LU2031234567</Code>
  <CodificationSystem>1</CodificationSystem>
  <InstrumentName>Europa Growth Fund R EUR ACC</InstrumentName>
```



```

<InstrumentCurrency>EUR</InstrumentCurrency>
<InstrumentPerformanceFee>N</InstrumentPerformanceFee>
<InstrumentDistributionCash>N</InstrumentDistributionCash>
<GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
<ProductType>U</ProductType>
<ManufacturerName>Europa Asset Management S.A.</ManufacturerName>
<ManufacturerLEI>529900T8BM49AURSD055</ManufacturerLEI>
<LeveragedOrContingentLiability>N</LeveragedOrContingentLiability>
<Fund>
  <FundShareClassWithoutRetrocession>N</FundShareClassWithoutRetrocession>
</Fund>
</GeneralInformation>

```

The `<Code>` plus `<CodificationSystem>` pair replaces a single ISIN field: the code is the actual identifier value, and the codification system is an integer enumeration telling the consumer what kind of code was used. 1 means ISO 6166 ISIN, 2 CUSIP, 3 SEDOL, 4 WKN, 5 Bloomberg Ticker, 8 FIGI, and so on, up to 99 for a producer-attributed internal code. The priority rule in the FinDatEx specification is: use ISIN when available, otherwise fall back to the codification in order of preference. For a European UCITS like the Europa Growth Fund, ISIN is always available.

`<ProductType>` uses a single-letter enumeration. U is UCITS, N is Non-UCITS (AIF), S is Structured Security, SF is Structured Fund, LM is UCITS Money Market Fund, NM is Non-UCITS Money Market Fund, ETC is Exchange Traded Commodity, B is Bond. The Europa Growth Fund is U.

The `<Fund>` sub-block at the bottom of `<GeneralInformation>` is selected via an `xs:choice` between `<Fund>` and `<StructuredSecurity>`. The schema offers the two alternatives because the fields that matter for a fund (retrocession status, exit-cost calculation basis) are different from the fields that matter for a structured security (notional-based versus item-based pricing, EUSIPA product category, quotation mode). For any UCITS or AIF, the `<Fund>` branch is chosen. The only mandatory field inside it is `<FundShareClassWithoutRetrocession>`: Y means the share class pays no inducement to distributors in the MiFID II sense, N means it does.

`<LeveragedOrContingentLiability>` is a yes/no flag used for MiFID II Article 62 reporting on leveraged instruments. For an ordinary long-only equity UCITS it is N; the flag is relevant for synthetic leveraged products or certain derivative-heavy funds.

The `<InstrumentPerformanceFee>` and `<InstrumentDistributionCash>` flags are derived from the fund's charter and its distribution policy: the Europa Growth Fund R-EUR-ACC class has no performance fee and is accumulating, so both values are N.

8.3.4 Target Market Fields

MiFID II defines the target market along five dimensions, and the FundsXML `<TargetMarket>` element maps each dimension to one nested child element with yes/no/neutral flags inside it. The five dimensions are `<InvestorType>`, `<KnowledgeAndExperience>`, `<AbilityToBearLosses>`, `<RiskTolerance>`, and `<ClientObjectives>`, plus a `<DistributionStrategy>` child that captures the MiFID II distribution strategy matrix.

For the Europa Growth Fund R-EUR-ACC, the block looks like this:

```
<TargetMarket>
  <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
  <InvestorType>
    <Retail>Y</Retail>
    <Professional>Y</Professional>
    <EligibleCounterparty>Y</EligibleCounterparty>
  </InvestorType>
  <KnowledgeAndExperience>
    <BasicInvestor>Y</BasicInvestor>
    <InformedInvestor>Y</InformedInvestor>
    <AdvancedInvestor>Y</AdvancedInvestor>
  </KnowledgeAndExperience>
  <AbilityToBearLosses>

<CompatibleWithClientsWhoCanNotBearCapitalLoss>N</CompatibleWithClientsWhoCanNotBearCapitalLoss>

<CompatibleWithClientsWhoDoNotNeedCapitalGuarantee>Y</CompatibleWithClientsWhoDoNotNeedCapitalGuarantee>

<CompatibleWithClientsWhoCanBearLossBeyondCapital>N</CompatibleWithClientsWhoCanBearLossBeyondCapital>
  </AbilityToBearLosses>
  <RiskTolerance>
    <PRIIPSMethodology>4</PRIIPSMethodology>
  </RiskTolerance>
  <ClientObjectives>
    <ReturnProfile>
      <ClientLookingForPreservation>N</ClientLookingForPreservation>
      <ClientLookingForCapitalGrowth>Y</ClientLookingForCapitalGrowth>
      <ClientLookingForIncome>N</ClientLookingForIncome>
    </ReturnProfile>
    <MinimumRecommendedHoldingPeriod>
      <Years>5</Years>
    </MinimumRecommendedHoldingPeriod>

<IntendedCompatibleWithClientsHavingSustainabilityPreferences>Y</IntendedCompatibleWithClientsHavingSustainabilityPreferences>
  </ClientObjectives>
  <DistributionStrategy>
    <ExecutionOnly>B</ExecutionOnly>
    <ExecutionWithCheckOrNonAdvisedServices>B</ExecutionWithCheckOrNonAdvisedServices>
    <InvestmentAdvice>B</InvestmentAdvice>
    <PortfolioManagement>B</PortfolioManagement>
  </DistributionStrategy>
</TargetMarket>
```

The values above reflect a UCITS equity fund that is appropriate for a broad range of retail and professional investors and is distributed through all four MiFID II distribution modes. A few notes on the enumerations:

- The `<InvestorType>/<Professional>` field takes `Y`, `N`, `P`, or `E`: `Y` covers both professional-per-se and elective-professional clients; `P` means only per-se professionals are compatible; `E` means only elective professionals are compatible; `N` means neither. The Europa Growth Fund accepts both, so the value is `Y`.
- All three knowledge levels are marked `Y` because a long-only European equity UCITS does not require any specialised financial sophistication. A more complex product (a leveraged ETF, a structured note, a convertible arbitrage fund) would typically mark `BasicInvestor` as `N` because the risk profile is too sophisticated for a first-time investor.
- The `<AbilityToBearLosses>` block uses three yes/no/neutral flags that together describe the loss profile. An equity UCITS can lose up to the entire invested capital (but no more — there is no leverage or margin), so the values are: *not* compatible with clients who cannot bear capital loss (`N`), compatible with clients who need no capital guarantee (`Y`), *not* compatible with clients who could face losses beyond their invested capital (`N` , because that scenario does not occur for a regular UCITS). The fourth optional field `<CompatibleWithClientsWhoCanBearLimitedCapitalLoss>` is omitted because it applies only to capital-protected or money market funds.
- `<RiskTolerance>/<PRIIPMethodology>` is the PRIIPs SRI on the 1-to-7 scale — the same number that also appears in the EPT (§8.4.3). The Europa Growth Fund's SRI is 4.
- `<ClientObjectives>/<ReturnProfile>` uses three flags for preservation, capital growth, and income. The Europa Growth Fund is a growth product, so only `<ClientLookingForCapitalGrowth>` is `Y`.
- The `<MinimumRecommendedHoldingPeriod>` is expressed as an `xs:choice` between `<Years>` (a decimal) and `<Category>` (a one-letter V/S/M/L/H code for Very short / Short / Medium / Long / Hold to maturity). For the Europa Growth Fund, the RHP is five years, so `<Years>5</Years>` is used.
- `<IntendedCompatibleWithClientsHavingSustainabilityPreferences>` is a yes-or-neutral flag (there is no "No" — the MiFID II sustainability preferences framework does not allow negative target market on this dimension). For the Europa Growth Fund, which is SFDR Article 8, the value is `Y`.
- `<DistributionStrategy>` uses a four-value enumeration per distribution channel: `R` (Retail only), `P` (Professional only), `B` (Both), `N` (Neither). The Europa Growth Fund is distributed through all channels to both segments, so every channel is marked `B`.

Table 8.1 — The MiFID II target market dimensions for the Europa Growth Fund R-EUR-ACC

Dimension	FundsXML element	Positive values	Notes
Client type	InvestorType	Retail=Y, Professional=Y, EligibleCounterparty=Y	All three categories
Knowledge and experience	KnowledgeAndExperience	Basic=Y, Informed=Y, Advanced=Y	Non-complex UCITS
Ability to bear losses	AbilityToBearLosses	Full loss acceptable	No capital guarantee
Risk tolerance	RiskTolerance/PRIIPS Methodology	4	PRIIPs SRI value
Client objectives	ClientObjectives/ReturnProfile	Capital growth only	Accumulating class

The institutional class I-EUR-DIST has a different target market: the `<Retail>` flag drops to `N`, because the 1,000,000 EUR minimum investment effectively excludes retail investors. The other dimensions remain the same, because the underlying portfolio and risk profile are identical across share classes. A producer emits one `<FinancialInstrument>` block per share class and repeats the identification, target-market, and cost content with the per-class differences.

8.3.5 Cost and Charges Fields

The cost fields are the most conceptually intricate part of EMT, because they implement the MiFID II cost classification, which is itself elaborate. MiFID II divides the forward-looking cost disclosure into several categories, and the FundsXML `<CostsAndChargesExAnte>/<Fund>` block mirrors those categories field by field:

- **Entry costs** — `<GrossMaxEntryCostNonAcquired>` (entry load not acquired to the fund), `<MaxEntryCostAcquired>` (entry load acquired to the fund), plus two optional Italy-specific variants.
- **Exit costs** — `<MaxExitCost>`, `<MaxExitCostAcquired>`, `<TypicalExitCost>`, plus further optional fields for structured funds.
- **Ongoing costs** — `<OngoingCosts>` (the headline figure, equivalent to the Ongoing Charges Figure from Chapter 5), `<ManagementFee>` (the pure management fee component, which must be less than or equal to the total ongoing costs), and the optional `<DistributionFee>`.
- **Transaction costs** — `<TransactionCosts>`.
- **Incidental costs** — `<IncidentalCosts>`, primarily capturing performance fees.

All of these fields are decimals expressed as proportions of NAV in annualised terms: `0.0175` means 1.75% per year. A `<CostsReferenceDate>` element closes the block, recording the reference date of the cost figures (which may or may not coincide with the general reference date).

For the Europa Growth Fund R-EUR-ACC class, the forward-looking cost block looks like this:

```

<CostsAndChargesExAnte>
  <Fund>
    <GrossMaxEntryCostNonAcquired>0.0500</GrossMaxEntryCostNonAcquired>
    <MaxExitCost>0.0000</MaxExitCost>
    <OngoingCosts>0.0175</OngoingCosts>
    <ManagementFee>0.0150</ManagementFee>
    <TransactionCosts>0.0012</TransactionCosts>
    <IncidentalCosts>0.0000</IncidentalCosts>
  </Fund>
  <CostsReferenceDate>2026-03-31</CostsReferenceDate>
</CostsAndChargesExAnte>

```

Reading the block: the share class allows a maximum entry load of 5% (the contractual cap in the prospectus), although actual distributor agreements typically waive or reduce this figure. The exit cost is zero. The ongoing cost is 1.75% per year, of which 1.50% is the management fee and the remaining 0.25% covers administration, depositary, and other ongoing costs. The transaction cost estimate is 0.12% per year, computed according to the FinDatEx methodology from the fund's realised trading activity over the preceding period. The incidental cost is zero because the fund charges no performance fee. The total of 1.87% (ongoing + transaction + incidental, excluding the prospectus-cap entry load) is the headline annual cost figure that a consumer would show to a retail investor.

The `<CostsAndChargesExPost>` block, populated only when the `<DataReportingExPost>` flag is set to `Y`, carries the realised costs for a past reference period. Its structure mirrors the ex-ante block but with slightly different field names (`EntryCostAcquired`, `OngoingCosts`, `ManagementFee`, `TransactionCosts`, `IncidentalCosts`) and records actually incurred rather than contractually possible charges. A producer that publishes ex-post figures typically does so on an annual basis rather than monthly, because the computations depend on a full reporting period's trading data.

An important relationship to Chapter 5: the cost structure in EMT is *not* identical to the `<Fees>` block in `FundStaticData`. The `<Fees>` block is the fund's own representation of its charges, organised by fee type and computation basis. The EMT cost block is the MiFID II representation, organised by MiFID II's own categorisation. The two must be *consistent* — the management fee in EMT must equal the management fee in `<Fees>`, the total ongoing costs must equal the Ongoing Charges Figure, the transaction-cost methodologies must align — but they are not byte-for-byte copies of each other. Producers who generate both representations must take care that they reconcile, because a downstream consumer comparing the two will flag any discrepancy.

An even more important relationship concerns EPT, which we will treat in §8.4.4. EPT (inside `<PRIIPS_V20>`) also carries cost fields, but with a PRIIPs-specific breakdown and with horizon-dependent values computed at one year and at the recommended holding period. The EMT figures above are annual percentages; the EPT figures express the same underlying cost structure over the RHP, and the arithmetic of amortising one-off costs over five years versus one year produces different aggregated numbers. The two representations are both authoritative, and consumers must use the one that matches their disclosure horizon.

8.3.6 EMT in Context — Multiple Share Classes

The `<EMT_V4_2>` element contains a sequence of `<FinancialInstrument>` children, so a fund with multiple share classes emits one block per class under the same EMT wrapper:

```

<EMT_V4_2>
  <FinancialInstrument>
    <!-- R-EUR-ACC: ISIN LU2031234567 -->
    ...
  </FinancialInstrument>
  <FinancialInstrument>
    <!-- R-CHF-ACC-HEDGED: ISIN LU2031234641 -->
    ...
  </FinancialInstrument>
  <FinancialInstrument>
    <!-- I-EUR-DIST: ISIN LU2031234724 -->
    ...
  </FinancialInstrument>
</EMT_V4_2>

```

A consumer iterates the `<FinancialInstrument>` children and matches each one to its corresponding share class by `<Code>`. Nothing in the schema prevents mixing share classes from different funds inside a single `<EMT_V4_2>` element (the element is a flat list), but operational practice is to keep one EMT block per fund per delivery for clarity. Where a single delivery covers many funds — typical for large multi-manager platforms — the `RegulatoryReportings/DirectReporting/EMT_V4_2` branch can contain one `<FinancialInstrument>` per share class across all funds, and consumers split them at processing time by the `<Code>` field.

The complete EMT example for the Europa Growth Fund R-EUR-ACC class — combining the `DataSetInformation`, `GeneralInformation`, `TargetMarket`, and `CostsAndChargesExAnte` sections from §§8.3.2–8.3.5 — appears in full in §8.9 alongside the other templates.

8.4 EPT — European PRIIPs Template

The PRIIPs-focused counterpart to EMT, embedded in FundsXML under `<IndirectReporting>/<PRIIPS_V20>`.

8.4.1 The PRIIPs Regulation and the KID

PRIIPs — the Packaged Retail and Insurance-based Investment Products regulation — has governed European retail investment disclosure since its original entry into force in 2018 and has applied to UCITS since January 2023, when the transition period expired and the UCITS-specific KIID regime gave way to the PRIIPs-wide KID. The PRIIPs framework covers not only traditional funds but also structured products, unit-linked insurance products, and similar retail-accessible instruments, and it aims to make disclosures comparable across these otherwise very different product types.

The central output of PRIIPs is the **KID** — the **Key Information Document** — a three-page standardised document that every retail investor must be given *before* they invest. The KID is highly structured and, in practice, very nearly identical in layout across all PRIIPs products. It must include: the product's name and identification, its risk level

(the SRI from 1 to 7), a set of performance scenarios over the recommended holding period, the costs the investor will bear, and a limited number of free-text sections that explain the product type and the intended target market.

The KID is a PDF that an end-investor reads. The *data* that populates the KID lives in a structured template — the **European PRIIPs Template, EPT** — which FinDatEx publishes and maintains. A manufacturer produces EPT; a distributor (or a specialised KID-generation service) consumes EPT and renders from it the final PDF document that goes into the investor's hands. EPT is the input; the KID is the output.

What the transition from UCITS KIID to PRIIPs KID changed:

- **Risk indicator:** the UCITS KIID used the SRRI (Synthetic Risk and Reward Indicator), a 1-to-7 scale based on past volatility of returns. The PRIIPs KID uses the SRI (Summary Risk Indicator), also 1 to 7, but computed as a combination of the Market Risk Measure (MRM, based on volatility and on the volatility-equivalent value) and the Credit Risk Measure (CRM). For most equity funds, CRM is 1 and the SRI is driven by MRM alone; for bond funds or structured products, the CRM contribution can dominate.
- **Performance presentation:** the UCITS KIID showed historical performance as a bar chart of the last ten years. The PRIIPs KID shows **four performance scenarios** — stress, unfavourable, moderate, favourable — as prospective outcomes over the recommended holding period, computed from the statistical distribution of the fund's historical returns.
- **Cost disclosure:** the UCITS KIID had a single Ongoing Charges figure. The PRIIPs KID has a detailed cost structure presented at multiple horizons (one year, half the RHP, and the full RHP), broken down into one-off, ongoing, and incidental categories.

EPT carries the structured data for all of these fields, and is therefore the template that any producer distributing UCITS to retail investors after January 2023 must emit alongside EMT.

8.4.2 Structure of the EPT Module

In FundsXML, EPT lives under `<IndirectReporting>/<PRIIPS_V20>`. The `PRIIPSType_V20` complex type has three children: a single `<EPTV2>` element that carries the core EPT payload, plus two optional `<CEPTV2History>` and `<CEPTV2Performance>` children that carry the Comfort EPT history and performance data (used primarily by insurance-linked products). For a plain UCITS like the Europa Growth Fund, only `<EPTV2>` is populated.

The `<EPTV2>` element groups the EPT fields into six mandatory sections:

1. `<DataSetInformation>` — the file-level metadata;
2. `<GeneralPortfolioInformation>` — product identification (the PRIIPs term "portfolio" refers to the *product* being disclosed, not to the fund's holdings portfolio from Chapter 6);
3. `<RiskAssessment>` — the SRI, MRM, CRM, and related indicators;
4. `<PerformanceScenario>` — the four PRIIPs performance scenarios;
5. `<Costs>` — the cost structure in the PRIIPs format;
6. `<Narratives>` — the standardised free-text fields that appear on the KID.

A skeleton:


```

<PRIIPS_V20>
  <EPTV2>
    <DataSetInformation>
      <Version>V21</Version>
      <!-- four Data_Reporting* flags -->
    </DataSetInformation>
    <GeneralPortfolioInformation>
      <!-- product identification – see §8.4.3 -->
    </GeneralPortfolioInformation>
    <RiskAssessment>
      <!-- SRI, MRM, CRM – see §8.4.4 -->
    </RiskAssessment>
    <PerformanceScenario>
      <!-- four scenarios – see §8.4.5 -->
    </PerformanceScenario>
    <Costs>
      <!-- cost structure – see §8.4.6 -->
    </Costs>
    <Narratives>
      <!-- KID free-text fields -->
    </Narratives>
  </EPTV2>
</PRIIPS_V20>

```

The `<Version>` element inside `<DataSetInformation>` takes the single value `V21`, reflecting EPT version 2.1 — the FinDatEx EPT revision that accompanies the PRIIPs technical standards currently in force. As with EMT, the element name (`PRIIPS_V20`) and the `<Version>` string (`V21`) refer to two different things: the former is the FundsXML-side XSD version, the latter is the FinDatEx-side template version.

One important schema quirk: the `DataSetInformation` block inside `<EPTV2>` uses yes/no fields of type `YesNoType`, which in this corner of the schema takes the string values `YES` and `NO` in full — not the single-letter `Y` and `N` used by EMT's `EMTYesNoType`. A producer who mechanically copies `Y / N` from one template to the other will hit a validation error. The fragments in this section use the correct `YES / NO` spelling.

8.4.3 General Portfolio Information

The `<GeneralPortfolioInformation>` block is the EPT equivalent of EMT's `<GeneralInformation>`: it identifies the product (manufacturer, LEI, group, the ISIN or internal code, the product name, currency, reference date, PRIIPs category, autocallable flag). For the Europa Growth Fund R-EUR-ACC class:

```

<GeneralPortfolioInformation>
  <PortfolioManufacturerName>Europa Asset Management S.A.</PortfolioManufacturerName>
  <PortfolioManufacturerGroupName>Europa Financial
Group</PortfolioManufacturerGroupName>
  <PortfolioManufacturerLEI>529900T8BM49AURSD055</PortfolioManufacturerLEI>
  <PortfolioID>
    <CodificationSystem>1</CodificationSystem>
    <Code>LU2031234567</Code>
  </PortfolioID>
</GeneralPortfolioInformation>

```



```

</PortfolioID>
<PortfolioName>Europa Growth Fund R EUR ACC</PortfolioName>
<PortfolioCurrency>EUR</PortfolioCurrency>
<GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
<PortfolioPRIIPSCategory>2</PortfolioPRIIPSCategory>
<IsAnAutocallableProduct>N0</IsAnAutocallableProduct>
</GeneralPortfolioInformation>

```

The `<PortfolioPRIIPSCategory>` field is a small integer on the closed enumeration 1 to 4, corresponding to the four PRIIPs product categories:

- **Category 1** — products where the retail investor could lose more than the amount invested (certain CFDs and derivative products) or where the product has no liquid market. Not applicable to traditional UCITS.
- **Category 2** — products whose return is linearly related to the performance of a diversified set of underlyings, and whose historical returns can be used directly for statistical modelling. Most long-only UCITS fall into Category 2. The Europa Growth Fund is Category 2.
- **Category 3** — products whose return depends non-linearly on the underlying (structured products with barriers, options, or path-dependency). These require Monte Carlo simulation for scenario generation.
- **Category 4** — products where the return depends on factors not observable in market data (with-profits insurance products, unit-linked wrappers with guarantees). These require their own scenario methodology.

The category choice drives much of the downstream scenario calculation, because the four categories use different methodologies for the performance scenario section. Most equity, bond, and balanced UCITS are Category 2; Category 3 is the natural home for structured funds.

8.4.4 Risk Assessment — SRI, MRM, and CRM

The `<RiskAssessment>` block carries the PRIIPs risk indicators and a handful of related fields. For the Europa Growth Fund:

```

<RiskAssessment>
  <ValuationFrequency>252</ValuationFrequency>
  <IS_Flexible>N0</IS_Flexible>
  <Existing_Credit_Risk>N0</Existing_Credit_Risk>
  <SRI>4</SRI>
  <IsSRIAdjusted>N0</IsSRIAdjusted>
  <MRM>4</MRM>
  <Recommended_Holding_Period>5</Recommended_Holding_Period>
  <HasAContractualMaturityDate>N0</HasAContractualMaturityDate>
  <Liquidity_Risk>L</Liquidity_Risk>
</RiskAssessment>

```

`<ValuationFrequency>` is the number of valuation days per year. The Europa Growth Fund values daily, so the value is 252 (business days). The field takes an enumeration — 1 (annual), 2 (biannual), 4 (quarterly), 12 (monthly), 24 (bimonthly), 52 (weekly), 104 (biweekly), 252 (daily) — and the producer chooses the one that matches the actual valuation cadence.

`<IS_Flexible>` marks whether the fund is a flexible fund under the PRIIPs Annex II clause 14. A flexible fund changes its asset-allocation profile over time and must disclose a historical VEV alongside a reference-allocation VEV. The Europa Growth Fund is a straight equity fund with a stable mandate and is therefore `NO`.

`<Existing_Credit_Risk>` is a yes/no flag. For a pure equity UCITS, the answer is `NO` because the fund holds no direct credit exposure. For a bond fund or a mixed fund with bond holdings, the answer is `YES` and the CRM (Credit Risk Measure) fields become load-bearing.

`<SRI>` is the headline 1-to-7 risk figure. For the Europa Growth Fund, the SRI is 4 — middle of the scale, consistent with a diversified European equity fund with moderate volatility and negligible credit risk.

`<MRM>` is the Market Risk Measure, also 1 to 7. The SRI is computed from the MRM and the CRM through a standardised matrix published in the PRIIPs regulatory technical standards; for a fund with no credit risk, $MRM = SRI$, and both are 4.

`<IsSRIAdjusted>` marks whether the manufacturer has exercised the regulatory option to increase the SRI beyond the value computed from the formula. This option exists for funds where the manufacturer believes the mechanical SRI understates the perceived investor risk — a rare occurrence, but the field is mandatory. For the Europa Growth Fund, the value is `NO`.

`<Recommended_Holding_Period>` is a decimal number of years, consistent with the EMT target market's `<MinimumRecommendedHoldingPeriod>/<Years>` field. Both say 5 for the Europa Growth Fund.

`<HasAContractualMaturityDate>` is `NO` for an open-ended UCITS and `YES` for a closed-ended or fixed-maturity product; the optional `<MaturityDate>` is populated only in the `YES` case.

`<Liquidity_Risk>` uses a single-letter enumeration: `M` (material liquidity risk), `I` (illiquid), or `L` (no liquidity issue). The Europa Growth Fund, invested in large-cap European equities, has no liquidity issue and is marked `L`.

8.4.5 Performance Scenarios

The `<PerformanceScenario>` block carries the four PRIIPs performance scenarios — **stress**, **unfavourable**, **moderate**, **favourable** — that appear on the KID. Each scenario is expressed as an annualised return (a decimal, e.g. `0.0540` for 5.40% per year) at the recommended holding period; the optional `<OneYear>` and `<HalfRHP>` children carry the shorter-horizon values.

For the Europa Growth Fund:

```
<PerformanceScenario>
  <ReturnUnfavorable>
    <RHP0rFirstCallDateOrFirstCallDate>-0.0120</RHP0rFirstCallDateOrFirstCallDate>
  </ReturnUnfavorable>
  <ReturnModerate>
    <RHP0rFirstCallDateOrFirstCallDate>0.0540</RHP0rFirstCallDateOrFirstCallDate>
  </ReturnModerate>
  <ReturnFavorable>
    <RHP0rFirstCallDateOrFirstCallDate>0.1080</RHP0rFirstCallDateOrFirstCallDate>
  </ReturnFavorable>
  <ReturnStress>
```

```

    <RHP0rFirstCallDate0rFirstCallDate>-0.2750</RHP0rFirstCallDate0rFirstCallDate>
  </ReturnStress>

  <PortfolioPastPerformanceDisclosureRequired>YES</PortfolioPastPerformanceDisclosureRequired>
</PerformanceScenario>

```

The child element name `<RHP0rFirstCallDate0rFirstCallDate>` is awkward — it is a schema artefact reflecting the fact that the same field is used for fixed-RHP products (where it carries the RHP value) and for auto-callable products (where it carries the value at the first call date). For a straight UCITS, the interpretation is always "at RHP".

The four values in turn. A moderate annual return of +5.40% per year over the five-year RHP represents the median of the historical-return distribution — the honest central estimate of what an investor might expect. The unfavourable scenario at -1.20% per year represents the 10th percentile of the distribution: a plausible but pessimistic outcome. The favourable scenario at +10.80% per year is the 90th percentile. The stress scenario at -27.50% per year is the extreme tail, derived from the worst historical episodes in the measurement window; this scenario is deliberately much worse than the unfavourable one to illustrate that equity investments can suffer severe drawdowns.

A KID generator reads these four values, compounds them over the RHP, applies them to the standard 10,000 EUR illustrative investment, and renders the resulting monetary outcomes on the KID performance table. For the Europa Growth Fund, the moderate scenario of +5.40% per year over five years produces an outcome of approximately 13,000 EUR on a 10,000 EUR initial investment — the figure a retail investor sees in the "What could I get in return?" section of the KID.

`<PortfolioPastPerformanceDisclosureRequired>` is `YES` for any fund that meets the PRIIPs Annex VIII conditions (a UCITS with at least one year of history), requiring the KID to link to the past performance document. For the Europa Growth Fund, which has been operating for several years, the flag is `YES`.

8.4.6 Costs in the PRIIPs Format

The `<Costs>` block decomposes the PRIIPs cost structure into three sub-blocks: `<OneOff>`, `<Ongoing>`, and `<Incidental>`. Each sub-block carries the cost components in the PRIIPs-specific breakdown:

```

<Costs>
  <OneOff>
    <EntryCost>0.0500</EntryCost>
    <EntryCostsAcquired>0.0000</EntryCostsAcquired>
    <ExitCostAtRHP>0.0000</ExitCostAtRHP>
    <SlidingExitCostIndicator>NO</SlidingExitCostIndicator>
  </OneOff>
  <Ongoing>
    <OtherCost>0.0025</OtherCost>
    <ManagementCosts>0.0150</ManagementCosts>
    <Transaction>0.0012</Transaction>
  </Ongoing>
  <Incidental>
    <ExistringIncidentalCostsPortfolio>NO</ExistringIncidentalCostsPortfolio>

```

```

    <ExistingIncidentalCostsPortfolio>NO</ExistingIncidentalCostsPortfolio>
  </Incidental>
</Costs>

```

A few points that the naive reader should be warned about. The `<Transaction>` field inside `<Ongoing>` carries the portfolio transaction costs — the same concept as EMT's `<TransactionCosts>` but at the same decimal-per-cent scale. The `<ManagementCosts>` field carries the management fee plus other administrative and operational running costs, which is a *superset* of the `<ManagementFee>` field in EMT's `<CostsAndChargesExAnte>/<Fund>`. A producer filling both templates must take care that the EPT's `<ManagementCosts>` is at least as large as the EMT's `<ManagementFee>`, and the difference, if any, reflects non-management administrative running costs that PRIIPs bundles into the management bucket but MiFID II breaks out separately.

The two similarly named children of `<Incidental>` — `<ExistringIncidentalCostsPortfolio>` and `<ExistingIncidentalCostsPortfolio>` — are not a typo in this book: they are two separate elements in the schema, and both are mandatory. The FinDatEx specification uses them to distinguish two types of incidental cost indicator, and the nearly identical spelling reflects a historical quirk of the template's field naming. A producer must populate both, and, in the absence of performance fees or carried interest, both are `NO`.

The `<Narratives>` section of `<EPTV2>` — a block of short free-text fields that populate the KID's narrative sections (intended target market description, investment objective, exit cost description, and so on) — is mandatory for all PRIIPs products. Its content is closed to a maximum character count and is intended to be reused verbatim on the final KID. For the Europa Growth Fund, the narrative fields are drawn from the fund's prospectus with minimal editorial adaptation to fit the character limits.

8.4.7 The EPT and EMT Cost Relationship

A critical practical point that is worth restating. EMT expresses cost figures as annual percentages, which is the MiFID II cost disclosure horizon. EPT expresses some cost figures over the recommended holding period as well — notably through the KID generator's downstream calculations, which aggregate the annual figures over the fund's RHP and display the result on the KID's "Costs over time" table. The input fields in `<PRIIPS_V20>` are still annualised decimals, but the downstream KID output amortises one-off costs across the RHP: for a 5% entry load and a 5-year RHP, the KID shows approximately 1% per year, and a 5-year investment horizon sees the full 5% taken at entry.

A consumer that reads "total cost of 1.87% per year" from EMT and "reduction in yield of 2.87% per year" from the KID generator for the same share class is not seeing a contradiction: the EMT number excludes the amortised entry load and the KID number includes it. Both figures are individually valid in their own disclosure contexts, and the difference reflects the different treatment of one-off costs under the two regulations.

8.5 EET — European ESG Template

The largest of the five templates by field count, and the most rapidly evolving of them. Embedded under `<DirectReporting>/<EET1.1.3>`.

8.5.1 SFDR and the Sustainability Regulatory Wave

SFDR — the Sustainable Finance Disclosure Regulation — is the regulatory cornerstone of European sustainable finance, and arguably the most consequential single piece of fund regulation of the last five years. It entered into force in March 2021, with its technical standards (the Level 2 RTS) becoming applicable in January 2023. SFDR sits inside a larger sustainability framework that includes the EU Taxonomy Regulation (which defines what counts as "environmentally sustainable" at the activity level), the Corporate Sustainability Reporting Directive (CSRD, which requires companies to publish sustainability data that fund managers can then consume), and a growing body of country-specific national regulations.

The mechanics of SFDR run primarily through three articles that classify the intensity of a fund's sustainability integration:

- **Article 6** — no specific ESG orientation. The fund discloses sustainability risks that are material to its investment decisions (which every fund must do), but it does not claim to pursue any particular sustainability outcome. This is the default classification for funds that do not incorporate ESG considerations beyond risk management.
- **Article 8** — "light green". The fund *promotes* environmental or social characteristics among its investment decisions, without making sustainability the overriding objective. Most European actively managed funds that market themselves as ESG-integrated, best-in-class, or exclusion-based fall into this category. Article 8 is a broad tent, and the precise definition of "promoting" characteristics is the subject of continuous clarification from ESMA and national regulators.
- **Article 9** — "dark green". The fund has *sustainable investment* as its specific objective, in the sense defined by SFDR Article 2(17). This is a narrower category, with stricter disclosure and allocation requirements. Article 9 funds must invest predominantly in assets that meet the SFDR definition of sustainable investment, and must apply the "do no significant harm" (DNSH) test to their holdings.

The **EU Taxonomy** is a separate but related framework that defines, at the level of individual economic activities, which ones qualify as environmentally sustainable. It currently covers climate change mitigation and climate change adaptation as its two primary environmental objectives, with four further objectives (water, circular economy, pollution prevention, biodiversity) progressively being added. A fund's Taxonomy alignment is the percentage of its portfolio investments whose underlying economic activities are Taxonomy-aligned, and it is reported separately from the SFDR classification.

Principal Adverse Impacts (PAI) are a list of fourteen mandatory indicators of negative sustainability impacts, published by the European Supervisory Authorities as part of the SFDR Level 2 RTS. Asset managers must disclose PAI values both at the entity level (aggregate across all their products) and, for Article 8 and Article 9 products, at the product level. The fourteen indicators cover greenhouse gas emissions, exposure to fossil fuels, biodiversity damage, water emissions, hazardous waste, violations of international norms, gender pay gaps, and exposure to controversial weapons.

The collective data burden of SFDR, the Taxonomy, and PAI is large, and FinDatEx addressed it with the **EET** — a template with several hundred fields that carries all the structured data points a European distributor or pre-contractual-disclosure generator needs.

8.5.2 Structure of the EET Module

EET is embedded in FundsXML at `<DirectReporting>/<EET1.1.3>`. The outer `EET1.1.3` element is the template-version container, of type `EETsType1.1.3`, which carries one or more `<EET1.1.3>` children of type `EETReportType1.1.3`. The inner-outer element reuse is a schema quirk — FinDatEx uses the same versioned string for the wrapper and for each report entry — but the sequence is one wrapper, one or more report entries. Each report entry corresponds to one share class (or one product at whatever level the template specification requires).

The `EETReportType1.1.3` type arranges the EET payload into several top-level sections:

1. `<DataSetInformation>` — an outer wrapper (the doubly-nested name is intentional in the schema) that contains an inner `<DataSetInformation>` with the file-level metadata and a `<ManufacturerInformation>` block;
2. `<GeneralFinancialInstrumentInformation>` — product identification (ISIN, name, currency);
3. `<MainCriterias>` — the top-level SFDR product-type classification;
4. `<MiFIDIDDTargetMarket>` — how the product fits into the MiFID II sustainability preferences framework;
5. `<ScreeningCriteria>` — exclusion and screening strategies;
6. `<TaxonomyAlignedInvestments>` — the EU Taxonomy alignment percentages (optional but populated for Article 8 and Article 9 funds);
7. `<PrincipalAdverseIndicators>` — the fourteen mandatory PAI values and the optional additional indicators (optional but populated for Article 8 and Article 9 funds that consider PAIs).

Groups 6 and 7 are by far the largest by field count, because they carry the structured numerical data that SFDR Level 2 mandates. The mandatory minimum is a surprisingly small subset of the full template — most of the numerical PAI and Taxonomy fields are optional, conditional on the fund's SFDR article and its PAI-consideration flag. An Article 6 fund that does not consider PAIs can produce a valid EET with only a handful of fields populated; an Article 8 fund with full PAI consideration and a Taxonomy disclosure is where the file grows to its full size.

The minimal mandatory structure of one `<EET1.1.3>` entry is surprisingly compact:

```
<EET1.1.3>
  <EET1.1.3>
    <DataSetInformation>
      <DataSetInformation>
        <Version>V1.1.3</Version>
        <Producer>
          <Name>Europa Asset Management S.A.</Name>
          <LEI>529900T8BM49AURSD055</LEI>
          <Email>eet@europa-am.lu</Email>
        </Producer>
        <FileGeneration>2026-03-31T18:30:00Z</FileGeneration>
        <DataReporting>
          <SFDRPreContractual>Y</SFDRPreContractual>
          <SFDRPeriodic>Y</SFDRPeriodic>
          <SFDREntityLevel>Y</SFDREntityLevel>
          <MiFID>Y</MiFID>
          <IDD>Y</IDD>
        </DataReporting>
```

```

    </DataSetInformation>
    <ManufacturerInformation>
      <Name>Europa Asset Management S.A.</Name>
      <CodeType>L</CodeType>
      <Code>529900T8BM49AURSD055</Code>
      <Email>esg@europa-am.lu</Email>
      <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
    </ManufacturerInformation>
    <ManufacturerWebsiteInformationStewardshipAndEngagement>https://www.europa-
am.lu/sustainability</ManufacturerWebsiteInformationStewardshipAndEngagement>
  </DataSetInformation>
  <GeneralFinancialInstrumentInformation>
    <IdentifyingData>LU2031234567</IdentifyingData>
    <TypeOfIdentification>1</TypeOfIdentification>
    <Name>Europa Growth Fund R EUR ACC</Name>
    <Currency>EUR</Currency>
  </GeneralFinancialInstrumentInformation>
  <MainCriterias>
    <SFDRProductType>8</SFDRProductType>
    <ConsidersPrincipleAdverseImpact>Y</ConsidersPrincipleAdverseImpact>
  </MainCriterias>
  <MiFIDIDDTargetMarket>

<EndClientSustainabilityPreferences_Considered>Y</EndClientSustainabilityPreferences_Co
nsidered>
  </MiFIDIDDTargetMarket>
  <ScreeningCriteria>
    <ExistingNegativeScreeningStrategy>Y</ExistingNegativeScreeningStrategy>
  </ScreeningCriteria>
</EET1.1.3>
</EET1.1.3>

```

Note the two features that are easy to stumble over. First, the double nesting of the version element name: the outer `<EET1.1.3>` is the wrapper, and each inner `<EET1.1.3>` is a single report entry. A file with several share classes emits several inner entries inside one outer wrapper. Second, the double nesting of `<DataSetInformation>` inside `<DataSetInformation>` — the outer is a section wrapper, the inner is the metadata block, and a `<ManufacturerInformation>` sibling of the inner `<DataSetInformation>` completes the outer section. The EET schema is, in this respect, more awkward than either EMT or EPT, and a producer building the block by hand benefits from keeping a validated template file as a copy source.

The `<DataReporting>` flags inside the inner `<DataSetInformation>` declare which regulatory contexts the EET instance supports. An `SFDRPreContractual=Y` value means the file is suitable to populate the SFDR pre-contractual disclosure; `SFDRPeriodic=Y` means it supports the annual periodic disclosure; `MiFID=Y` and `IDD=Y` mean the file carries the data needed for the MiFID II / IDD sustainability preferences assessment. The Europa Growth Fund supports all five regulatory uses.

8.5.3 SFDR Article Classification

The entry point for any EET consumer is the single field that determines which version of the disclosure the fund must produce: **under which SFDR article is the fund classified?**

The `<MainCriteria>/<SFDRProductType>` field carries the classification as a single-digit string: `0` for "not in SFDR scope" (some structured products and non-EU AIFs), `6` for Article 6, `8` for Article 8, or `9` for Article 9. The **Europa Growth Fund is an Article 8 fund**, so `<SFDRProductType>8</SFDRProductType>`.

The Europa Growth Fund promotes environmental and social characteristics through the integration of ESG criteria into its active investment process — specifically, it applies a best-in-class screen that tilts the portfolio towards companies with stronger ESG profiles relative to their sector peers, and it excludes companies in controversial sectors (tobacco, controversial weapons, thermal coal extraction above a certain threshold). It does not, however, have "sustainable investment" as its overriding objective in the Article 9 sense — the fund's primary investment objective remains the long-term capital appreciation of its investors through a diversified European equity portfolio, and the ESG integration is a *means* rather than the *end*.

The `<ConsidersPrincipleAdverseImpact>` flag alongside `<SFDRProductType>` is the second-most important field in the EET: it declares whether the fund considers Principal Adverse Impacts at the *product* level. For Article 8 and Article 9 products, setting this flag to `Y` opens the door to a much larger PAI section in the template; setting it to `N` tells the consumer that the fund does not report PAI at the product level and that the optional PAI section will be absent. The Europa Growth Fund considers PAIs, so the flag is `Y`.

Article 8 classification brings concrete disclosure obligations that the EET must carry:

- **Which environmental or social characteristics does the fund promote?** For the Europa Growth Fund: reduction of carbon intensity relative to the benchmark (MSCI Europe), avoidance of companies with major ESG controversies, and a tilt towards companies with diverse boards.
- **What methodology is used to measure and demonstrate the promotion?** For the Europa Growth Fund: a proprietary ESG scoring methodology that combines data from MSCI ESG, Sustainalytics, and the fund's own analyst assessments, applied at the security level in the portfolio construction process.
- **What minimum share of investments is sustainable within the meaning of SFDR?** For the Europa Growth Fund: 30% of the portfolio is committed to SFDR-sustainable investments, defined as companies whose primary economic activity meets the DNSH criteria and whose operations comply with minimum safeguards.
- **Which principal adverse impacts does the fund consider at the product level?** See §8.5.4.

All of these questions map to specific optional fields in the EET schema — each with its own FinDatEx field number (20200, 20210, 20220, and so on) and its own constraint on when it must be populated. A producer that fills the EET correctly for an Article 8 fund populates a substantial subset of the optional fields, even though none of them are schema-mandatory. A consumer that receives an Article 8 EET with only the five mandatory fields filled in would accept the file as schema-valid but flag it as semantically incomplete.

Article 9 funds would have additional and stricter disclosures: a description of the specific sustainability objective being pursued, the binding elements of the investment strategy, the reference benchmark (for passive Article 9 funds), and the methodology to demonstrate alignment with the Paris Agreement or other relevant international frameworks. The Europa Growth Fund, as an Article 8 fund, does not populate these fields; they appear in the EET schema but are left empty for Article 8 classifications.

8.5.4 PAI Fields and Taxonomy Alignment

The fourteen mandatory PAI indicators live inside the optional `<PrincipalAdverseIndicators>` section of `EETReportType1.1.3`, organised into subgroups by theme: `ClimateAndEnvironmentRelatedIndicators` (greenhouse gas emissions, carbon footprint, energy, biodiversity, water, waste) and `SocialAndEmployeeMatters` (UN Global Compact violations, gender pay gap, board diversity, controversial weapons).

The fourteen mandatory PAI indicators divide into two groups.

Environmental PAIs (nine indicators):

1. GHG emissions (Scope 1, Scope 2, Scope 3, and total) of investee companies
2. Carbon footprint of the portfolio
3. GHG intensity of investee companies
4. Exposure to companies active in the fossil fuel sector
5. Share of non-renewable energy consumption and production of investee companies
6. Energy consumption intensity per high impact climate sector
7. Activities negatively affecting biodiversity-sensitive areas
8. Emissions to water
9. Hazardous waste and radioactive waste ratio

Social and employee-related PAIs (five indicators):

1. Violations of the UN Global Compact principles and the OECD Guidelines for Multinational Enterprises
2. Lack of processes and compliance mechanisms to monitor compliance with those principles
3. Unadjusted gender pay gap of investee companies
4. Board gender diversity of investee companies
5. Exposure to controversial weapons (anti-personnel mines, cluster munitions, chemical weapons, biological weapons)

Each PAI is quantified either as a numerical value (for indicators with a natural numeric measure, such as GHG emissions or carbon intensity) or as a percentage exposure (for indicators that are effectively binary at the company level, such as controversial weapons exposure). The schema carries each indicator as a small structured type with sub-fields for value, strategy, and coverage — capturing the fact that a reported PAI value is only as reliable as the share of the portfolio for which underlying data was available.

A critical and practically difficult point: **not all portfolio companies report PAI data**. Asset managers must fill the gaps in their PAI coverage through a combination of issuer-reported data (where available), third-party data provider estimates (MSCI ESG, Sustainalytics, ISS ESG, Moody's ESG are the main providers), and modelled values from industry benchmarks. The EET distinguishes between measured and estimated values through dedicated flags, so that consumers can assess the data quality of each indicator. A PAI value marked as "estimated" is weaker evidence than a value marked as "reported", and a downstream consumer (for example, a sustainability-focused distributor) may treat the two differently in its own product evaluation.

Taxonomy Alignment is reported separately from PAI, inside the optional `<TaxonomyAlignedInvestments>` section, and is expressed as the percentage share of the portfolio whose underlying investments are Taxonomy-aligned. Fields include the minimum committed alignment (separately including and excluding sovereign bonds), the methodology used for the calculation (revenue-based, CapEx-based, or OpEx-based), the share of transitional and enabling activities, and the breakdown by the six environmental objectives. The calculation runs from the investee company's reported turnover, capital expenditure, or operational expenditure that comes from Taxonomy-eligible economic activities. For a European equity fund like the Europa Growth Fund, Taxonomy alignment typically falls in the range of 15% to 35%, depending on the sector allocation and on how many of the portfolio companies have begun to report their Taxonomy alignment (CSRD is progressively expanding the reporting population).

8.5.5 The EET in Practice

For the March 2026 delivery of the Europa Growth Fund, the EET instance contains the mandatory skeleton shown in §8.5.2 plus a populated `<TaxonomyAlignedInvestments>` section and a populated `<PrincipalAdverseIndicators>` section — together amounting to several dozen fields. The full instance is carried in Appendix D as part of the complete sample file. The abbreviated version in §8.9 below — showing only the mandatory skeleton — is the shape of the EET entry, with the understanding that a production-grade Article 8 fund emits substantially more content than the bare minimum.

8.6 EFT — European Feedback Template

The lightest of the five templates, and the one that runs in the reverse direction — from distributor to manufacturer rather than from manufacturer to distributor. Embedded under `<DirectReporting>/<EFTs>`.

8.6.1 What EFT Is and Why It Exists

MiFID II does not treat the manufacturer-distributor relationship as a one-way street. Beyond the manufacturer's obligation to deliver EMT, EPT, and EET *to* the distributor, there is a corresponding obligation on the distributor to provide **feedback** *back* to the manufacturer. The feedback covers questions that only the distributor can answer: how is the product selling in practice, do the actual buyers match the manufacturer's target market, are there client complaints, has the distributor identified any product governance issues?

The obligation sits within MiFID II's product governance framework — specifically, under the requirement that manufacturers and distributors work together to ensure ongoing product suitability. Without systematic feedback from distributors, a manufacturer cannot know whether its product is being sold to the clients it was designed for, and cannot update its target market definition or product governance arrangements in response to real-world evidence.

EFT is the FinDatEx template that standardises this feedback. It is created by the **distributor** and sent to the **manufacturer**, typically once a year (the minimum frequency under MiFID II) but sometimes more often when the manufacturer wants quarterly data. The content covers distribution statistics, target market alignment, and product governance observations.

The key operational consequence of the reverse direction is that **EFT does not appear in a fund's own monthly delivery**. The Europa Growth Fund's 31 March 2026 delivery flows from Europa Asset Management S.A. to its distributors, and it carries EMT, EPT, EET, and TPT but not EFT, because the fund's manufacturer is not the party producing feedback. EFT would appear in a separate delivery flowing from each of the fund's distributors *back* to Europa Asset Management S.A., on the distributor's own schedule. §8.9 will reflect this absence in the complete example block.

8.6.2 Structure of the EFT Module

The FundsXML embedding follows the same wrapper pattern as the other templates. The `<EFTs>` element (of type `EFTType`) is the container, and each child `<EFT>` (of type `EFTReportType`) is one feedback report. A report has three mandatory sections:

1. `<DataSetInformation>` — the file-level metadata, split into `ReportInformationAndScope` (version, generation timestamp, reporting period, and the `M / D / B` reference target market flag for whether the deviations are measured against the Manufacturer's, the Distributor's, or Both target markets), `SubmitterEntityInformation` (the distributor's name, identifier, identifier type, and position in the distribution chain), and `ManufacturerEntityInformation` (the recipient manufacturer's equivalent fields);
2. `<GeneralFinancialInstrumentInformation>` — product identification (ISIN, identification type, name, and total number of transactions for the period);
3. optional `<DeviationReportManufacturerTargetMarketPerspective>` and `<DeviationReportDistributorTargetMarketPerspective>` blocks — the actual feedback data, broken down into `Sales_OTM_NTM` (sales outside target market or in negative target market) and `Widening` sub-blocks.

The mandatory minimum is small. A distributor that observed no target-market deviations during the reporting period can emit a valid EFT with only the three mandatory sections populated and no deviation reports — effectively a "nil return" that communicates "we distributed the product, we reviewed the target market alignment, we observed no issues".

A complete minimal EFT entry:

```
<EFTs>
  <EFT>
    <DataSetInformation>
      <ReportInformationAndScope>
        <Version>V1</Version>
        <FileGeneration>2026-04-05T08:00:00Z</FileGeneration>
        <PeriodStartData>2026-01-01</PeriodStartData>
        <PeriodEndData>2026-03-31</PeriodEndData>
        <ReferenceTargetMarket>B</ReferenceTargetMarket>
      </ReportInformationAndScope>
      <SubmitterEntityInformation>
        <Name>Nordbank Privatkunden AG</Name>
        <Identifier>5299001NORDBANK00001</Identifier>
        <IdentifierType>L</IdentifierType>
        <PositionInTheDistributionChain>D</PositionInTheDistributionChain>
```

```

</SubmitterEntityInformation>
<ManufacturerEntityInformation>
  <Name>Europa Asset Management S.A.</Name>
  <Identifier>529900T8BM49AURSD055</Identifier>
  <IdentifierType>L</IdentifierType>
</ManufacturerEntityInformation>
</DataSetInformation>
<GeneralFinancialInstrumentInformation>
  <IdentifyingData>LU2031234567</IdentifyingData>
  <TypeOfIdentification>1</TypeOfIdentification>
  <Name>Europa Growth Fund R EUR ACC</Name>
  <TotalNumberOfTransactions>1243</TotalNumberOfTransactions>
</GeneralFinancialInstrumentInformation>
<DeviationReportManufacturerTargetMarketPerspective>
  <Sales_OTM_NTM>
    <RetailInvestorTypeAndSelfService>0</RetailInvestorTypeAndSelfService>

<RetailInvestorTypeAndExecutionWithSuitabilityTest>0</RetailInvestorTypeAndExecutionWithSuitabilityTest>
    <KnowledgeAndExperienceAndSelfService>2</KnowledgeAndExperienceAndSelfService>

<KnowledgeAndExperienceAndWithSuitabilityTest>0</KnowledgeAndExperienceAndWithSuitabilityTest>
  </Sales_OTM_NTM>
</DeviationReportManufacturerTargetMarketPerspective>
</EFT>
</EFTs>

```

Reading the block: a hypothetical distributor (Nordbank Privatkunden AG) reports on the Q1 2026 period. The reference target market is **B** (Both) — the distributor checked deviations against both the manufacturer's target market and its own refined distributor target market. The distributor processed 1,243 transactions in the R-EUR-ACC share class during the period. The deviation report shows that all 1,243 transactions were aligned on the Retail Investor Type dimension (0 deviations) but that 2 transactions were identified as deviations on the Knowledge and Experience dimension under self-service execution — presumably, two clients with "Basic" knowledge level bought the fund via the bank's online execution-only channel despite the product's target market being Informed / Advanced. The distributor is required to disclose this, and the manufacturer will read it during the annual product governance review.

This feedback, once processed by Europa Asset Management S.A., becomes one of the inputs to the fund's annual product governance review. If several distributors were reporting consistent misalignment on one of the target market dimensions, the manufacturer would be expected to investigate — perhaps the product's risk characteristics have drifted, or the target market definition was drawn too narrowly for the actual investor base.

For the purpose of this chapter, the important point is that EFT exists, runs in the reverse direction, and is absent from the 31 March 2026 delivery of the Europa Growth Fund produced by Europa Asset Management S.A. The next section, on TPT, returns to a template that *is* present in the delivery.

8.7 TPT — Tripartite Template for Solvency II

The template that gives insurance investors the data they need to hold fund positions under Solvency II. Embedded under `<IndirectReporting>/<TripartiteTemplateSolvencyII_V6>`.

8.7.1 Solvency II and Insurance Reporting

Solvency II is the European insurance regulation that has governed the capital adequacy of insurance companies since 2016. Its central mechanism is the **Solvency Capital Requirement (SCR)** — a risk-based capital charge calculated from the insurance company's asset and liability positions, designed to ensure that the insurer can survive a one-in-two-hundred-year adverse event over a one-year horizon. The SCR is updated at least annually and is reported to the national supervisor through the Quantitative Reporting Templates (QRTs) that Solvency II specifies.

Many European insurance companies hold fund positions as part of their investment portfolios. When a life insurer invests 50 million EUR in the Europa Growth Fund through an insurance mandate or a unit-linked wrapper, the Europa Growth Fund position becomes part of the insurer's asset base, and the insurer must calculate its SCR including that position. Solvency II requires this calculation to be done on a **lookthrough basis** — not by treating the fund as a single opaque asset with an estimated overall risk, but by looking through to the fund's underlying holdings (the individual European equities, cash positions, and currency forwards we met in Chapter 6) and applying the Solvency II risk modules to each underlying.

The problem is that the insurer does not have direct access to the fund's portfolio detail. Without a standardised data exchange, the insurer would have to approximate the lookthrough data or negotiate bilateral data-sharing arrangements with each fund manager it held positions with. The operational burden would be enormous.

TPT — the Tripartite Template — is the FinDatEx solution. The name comes from the three parties historically involved in defining the template: asset managers (who produce fund data), fund administrators (who hold the position details), and insurers (who need the data for Solvency II). TPT standardises the lookthrough data into a single well-defined format that every European insurer knows how to consume. An asset manager that produces TPT for each of its funds makes those funds "Solvency II ready" for any insurance investor, without needing bilateral negotiation.

TPT has become operationally critical for any fund that wants to be held by insurance money. The Europa Growth Fund carries insurance investors among its I-EUR-DIST class unit holders, and Europa Asset Management S.A. therefore emits TPT in every monthly delivery of the fund's data.

8.7.2 Structure of TPT V6

In FundsXML, TPT lives at `<IndirectReporting>/<TripartiteTemplateSolvencyII_V6>`. The element name carries the version number explicitly: `V6` is FinDatEx Tripartite Template version 6.0 of March 2022, which accompanies the current Solvency II technical standards. The earlier versions (`TripartiteTemplateSolvencyII`, `TripartiteTemplateSolvencyII_V5`) are still in the schema for backward compatibility.

The type `TripartiteTemplateSolvencyIIType_V6` is a thin wrapper whose content is a sequence of `<Portfolio>` children, each of type `SolvencyIIPortfolioType_V6`. The term "portfolio" here follows the Solvency II convention — one `<Portfolio>` corresponds to one fund or one share class — not the FundsXML

convention where `<Portfolio>` refers to a holdings list inside `FundDynamicData`. The two uses of the same word are a source of confusion; within this section the Solvency II meaning prevails.

A `<Portfolio>` has the following mandatory fields at the top level:

- `<TPTVersion>` — must be exactly the string `V6.0`. This is a regulatory versioning marker inside the schema, distinct from the element name.
- `<PortfolioID>` — a `SIISecurityCodificationType_V6` pair (CodificationSystem integer plus Code string) identifying the fund.
- `<PortfolioName>` — the fund's name.
- `<PortfolioCurrency>` — an ISO 4217 currency code.
- `<TotalNetAssets>` — the fund's total net asset value in portfolio currency.
- `<ValuationDate>` — the date the portfolio inventory is valid for.
- `<ReportingDate>` — the date the data refers to (typically the end of the reporting period).
- `<CompleteSCRDelivery>` — `Y` if the producer has computed all SCR contributions for each position and included them in the file; `N` otherwise. The distinction matters because a `Y` file lets the insurance consumer use the pre-computed SCR inputs directly, whereas an `N` file requires the insurer to compute its own SCR from the raw positions.
- `<Positions>` — the holdings lookthrough, containing one or more `<Position>` children of type `SIIPositionType_V6`.

Each position inside `<Positions>` is itself a complex block with mandatory sub-fields including `InstrumentCIC` (the four-character CIC code classifying the instrument type and country), `InstrumentCode` (the ISIN or alternative identifier), `InstrumentName`, `Valuation` (with six mandatory monetary sub-fields in quotation currency and portfolio currency plus market-exposure figures), and `AdditionalInformation` (a structured block of optional QRT-specific sub-fields).

A position for a single equity holding — SAP SE — looks like this:

```
<Position>
  <InstrumentCIC>DE31</InstrumentCIC>
  <InstrumentCode>
    <CodificationSystem>1</CodificationSystem>
    <Code>DE0007164600</Code>
  </InstrumentCode>
  <InstrumentName>SAP SE</InstrumentName>
  <Valuation>
    <QuotationCurrency>EUR</QuotationCurrency>
    <MarketValueQC>7450000.00</MarketValueQC>
    <CleanValueQC>7450000.00</CleanValueQC>
    <MarketValuePC>7450000.00</MarketValuePC>
    <CleanValuePC>7450000.00</CleanValuePC>
    <PositionWeight>0.0300</PositionWeight>
    <MarketExposureQC>7450000.00</MarketExposureQC>
    <MarketExposurePC>7450000.00</MarketExposurePC>
    <MarketExposureWeight>0.0300</MarketExposureWeight>
  </Valuation>
```

```
<AdditionalInformation/>
</Position>
```

The six mandatory monetary fields — `MarketValueQC` , `CleanValueQC` , `MarketValuePC` , `CleanValuePC` , `MarketExposureQC` , `MarketExposurePC` — carry the position's market value and market exposure in both quotation currency (QC, the currency the instrument is quoted in) and portfolio currency (PC, the fund's base currency). For an equity holding in a single-currency fund, the QC and PC values are identical, and the clean values equal the market values (there is no accrued interest on equity). For a foreign-currency equity or for a bond with accrued coupon, the QC and PC values differ and the clean and market values differ respectively. The `PositionWeight` expresses the share of the fund's net assets that the position represents, as a decimal (`0.0300` = 3% of NAV).

The four-character **CIC (Complementary Identification Code)** is an EIOPA-defined classification standardised in the Solvency II regulation. Its format is `CC` (ISO country code) followed by two digits, where the first digit identifies the asset category (1 = government bond, 2 = corporate bond, 3 = listed equity, 4 = unlisted equity, and so on) and the second digit refines it. `DE31` means "listed equity in Germany, asset sub-category 1"; `FR31` would be the French equivalent, `CH31` the Swiss equivalent. The CIC is the single most important classification field in TPT, because the Solvency II SCR calculation uses it as the primary key for applying the standard-formula shock factors — 39% for EEA listed equities, 49% for non-EEA listed equities, different percentages for different bond categories.

`<AdditionalInformation>` is mandatory but may be empty (as in the example), because all of its children are optional. Its optional children cover the fund custodian name, infrastructure investment flags, Type 1 private equity eligibility, counterparty sector codes, collateral eligibility, and collateral market value — fields that are relevant only for specific asset categories.

8.7.3 TPT in Practice

The **holdings lookthrough** section of TPT is by far the largest part of the template by row count: a fund with 150 equity positions plus three cash balances plus two FX forwards produces 155 `<Position>` entries, each with ten to twenty sub-fields. A full delivery of the Europa Growth Fund's TPT is therefore several thousand lines of XML, which is why the example in §8.9 shows only a single representative position.

The TPT lookthrough exists *in addition to* the native `<Portfolios>` block in `FundDynamicData` (Chapter 6). The two representations of the portfolio are not the same — they serve different consumers and use different field formats — but they describe the same economic reality. A consumer that reads both should find that the position list and the aggregate market values reconcile, within small rounding tolerances. Producers must maintain consistency between the two views: a mismatch between the native portfolio and the TPT holdings section is a data quality error that a validating consumer should flag.

An important decision for TPT producers: **should the holdings lookthrough be complete, or should it be aggregated?** The standard answer for Solvency II is *complete* — every individual holding should appear in TPT, not just sector or country aggregates. The reason is that precise SCR calculations depend on the characteristics of individual issuers (their rating, their country of risk, their sector) and aggregation would lose the information that the calculation needs. The cost of the full lookthrough is a much larger TPT section per fund, and deliveries covering many funds can reach tens or hundreds of thousands of TPT position rows. Producers who emit batched deliveries need to watch the file sizes and, where necessary, split batches into smaller chunks.

The Europa Growth Fund's TPT for 31 March 2026 carries the full 155-position lookthrough, with `<CompleteSCRDelivery>` set to `N` (the manufacturer provides the raw positions but leaves the SCR computation to each consuming insurer, which is the more common pattern because insurers prefer to run the calculation through their own internal models). An insurance consumer reads the block, runs its own SCR calculation, and determines the capital charge that the fund position adds to the insurer's overall SCR. The entire workflow is repeatable, auditable, and standardised — exactly what Solvency II requires.

8.8 ESAP — European Single Access Point

The final regulatory topic of the chapter is not a FinDatEx template but an EU infrastructure initiative that affects how FundsXML deliveries reach their ultimate consumers.

8.8.1 What ESAP Is

The **European Single Access Point** is an EU-wide regulatory disclosure portal whose phased rollout began in 2026. Its purpose is to provide a single, centralised point through which investors, analysts, regulators, and the general public can retrieve public financial and sustainability disclosures from across the European financial sector. Before ESAP, these disclosures were scattered across national regulators, company websites, industry databases, and private data vendors, with no common format, no common access protocol, and no way to aggregate them across borders without bilateral data integration.

ESAP is part of the **Capital Markets Union** (CMU), the longer-running EU initiative to reduce the fragmentation of European capital markets. The CMU goals include making it easier for companies to raise capital across borders, for investors to access cross-border investment opportunities, and for retail savers to compare products from different countries. ESAP supports the last of these goals by ensuring that the regulatory data needed for comparison is available in one place, in structured machine-readable formats.

For funds specifically, ESAP aggregates several kinds of regulatory disclosure: SFDR pre-contractual and periodic disclosures (the Article 6/8/9 information that every fund must provide under the sustainable finance regulation); PRIIPs KIDs (for retail products); prospectuses for regulated investment products; and sustainability reports under CSRD. Not every fund disclosure goes to ESAP — bilateral deliveries (a monthly FundsXML file from Europa Asset Management S.A. to Nordbank Privatkunden AG, for example) are private contractual exchanges and are outside ESAP's remit. What ESAP covers is the category of disclosures that are supposed to be publicly available under their respective regulations.

8.8.2 FundsXML and ESAP

ESAP is a consumer of regulatory disclosures rather than an element of the FundsXML schema. As of the current FundsXML 4.2.8 release there is **no dedicated ESAP metadata element inside `ControlData` or elsewhere in the schema**, and no schema-level mechanism that declares "this file is destined for ESAP". That is a conscious choice: ESAP's submission channel is a separate pipeline governed by ESMA-maintained specifications, and the FundsXML association has so far held back from adding ESAP-specific elements until the submission format stabilises across the phased rollout.

In practice, a producer that publishes to ESAP typically does so by transforming an existing FundsXML document (or a reg-reporting-only projection of it) into whatever format the ESAP gateway currently accepts for the category in question — which, depending on the category, may be XBRL, ESMA-specified XML, or increasingly a FundsXML subset aligned with the relevant FinDatEx template. The routing, the authentication, and the acceptance rules all live outside the FundsXML schema, in the pipeline code that sits between the producer's systems and the ESAP API.

A brief list of practical consequences for a fund data pipeline in 2026:

- **Private bilateral deliveries remain private.** The monthly FundsXML file from a manufacturer to one of its distributors is not automatically published to ESAP; it is a private contractual exchange.
- **SFDR disclosures are the main ESAP category that touches a typical fund pipeline today.** A manufacturer's quarterly SFDR pre-contractual disclosure — which overlaps substantially with the content of the EET in `DirectReporting` — is the document that goes to ESAP, typically through a separate submission channel.
- **Expect the situation to evolve.** Each FundsXML minor release since 4.2 has added hooks or allowed extensions for progressively more ESAP-oriented metadata, and future versions may well introduce dedicated ESAP elements. Chapter 10 will return to the operational pattern of publishing to ESAP as part of a production pipeline.

For the Europa Growth Fund, the 31 March 2026 delivery is a private bilateral delivery — from Europa Asset Management S.A. to one of its distributors — and does not go through ESAP. A separate quarterly SFDR disclosure, due at the end of Q1, is the artefact that reaches ESAP. The two artefacts share some underlying data but are routed, packaged, and authenticated differently.

8.9 The Complete RegulatoryReportings Block for the Europa Growth Fund

The integrative close of the chapter. A `<RegulatoryReportings>` block containing four of the five templates — EMT, EFT-free, EET, TPT, and PRIIPs — for the 31 March 2026 delivery of the Europa Growth Fund R-EUR-ACC share class.

EFT is absent, because this delivery flows from the manufacturer to distributors rather than in the reverse direction. The absence is structural and correct, and the block below reflects it. The four modules that *are* present — `EMT_V4_2`, `EET1.1.3`, `TripartiteTemplateSolvencyII_V6`, and `PRIIPS_V20` — have been introduced in the preceding sections; here we see them side by side in one container. Because the embedding uses the two branches `DirectReporting` and `IndirectReporting`, the templates appear in their respective branches rather than as flat siblings.

The abbreviated block is:

```
<RegulatoryReportings>

  <DirectReporting>

    <!-- EMT: MiFID II costs and target market -->
    <EMT_V4_2>
      <FinancialInstrument>
```

```

<DataSetInformation>
  <Version>V4</Version>
  <ProducerName>Europa Asset Management S.A.</ProducerName>
  <ProducerLEI>529900T8BM49AURSD055</ProducerLEI>
  <ProducerEmail>emt@europa-am.lu</ProducerEmail>
  <FileGenerationDateTime>2026-03-31T18:00:00Z</FileGenerationDateTime>
  <DataReportingTargetMarket>Y</DataReportingTargetMarket>
  <DataReportingExAnte>Y</DataReportingExAnte>
  <DataReportingExPost>N</DataReportingExPost>
</DataSetInformation>
<GeneralInformation>
  <Code>LU2031234567</Code>
  <CodificationSystem>1</CodificationSystem>
  <InstrumentName>Europa Growth Fund R EUR ACC</InstrumentName>
  <InstrumentCurrency>EUR</InstrumentCurrency>
  <InstrumentPerformanceFee>N</InstrumentPerformanceFee>
  <InstrumentDistributionCash>N</InstrumentDistributionCash>
  <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
  <ProductType>U</ProductType>
  <ManufacturerName>Europa Asset Management S.A.</ManufacturerName>
  <ManufacturerLEI>529900T8BM49AURSD055</ManufacturerLEI>
  <LeveragedOrContingentLiability>N</LeveragedOrContingentLiability>
  <Fund>
    <FundShareClassWithoutRetrocession>N</FundShareClassWithoutRetrocession>
  </Fund>
</GeneralInformation>
<TargetMarket>
  <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
  <InvestorType>
    <Retail>Y</Retail>
    <Professional>Y</Professional>
    <EligibleCounterparty>Y</EligibleCounterparty>
  </InvestorType>
  <KnowledgeAndExperience>
    <BasicInvestor>Y</BasicInvestor>
    <InformedInvestor>Y</InformedInvestor>
    <AdvancedInvestor>Y</AdvancedInvestor>
  </KnowledgeAndExperience>
  <AbilityToBearLosses>

<CompatibleWithClientsWhoCanNotBearCapitalLoss>N</CompatibleWithClientsWhoCanNotBearCapitalLoss>

<CompatibleWithClientsWhoDoNotNeedCapitalGuarantee>Y</CompatibleWithClientsWhoDoNotNeedCapitalGuarantee>

<CompatibleWithClientsWhoCanBearLossBeyondCapital>N</CompatibleWithClientsWhoCanBearLossBeyondCapital>
  </AbilityToBearLosses>
  <RiskTolerance>
    <PRIIPSMethodology>4</PRIIPSMethodology>
  </RiskTolerance>

```

```

    <ClientObjectives>
      <ReturnProfile>
        <ClientLookingForPreservation>N</ClientLookingForPreservation>
        <ClientLookingForCapitalGrowth>Y</ClientLookingForCapitalGrowth>
        <ClientLookingForIncome>N</ClientLookingForIncome>
      </ReturnProfile>
      <MinimumRecommendedHoldingPeriod>
        <Years>5</Years>
      </MinimumRecommendedHoldingPeriod>

<IntendedCompatibleWithClientsHavingSustainabilityPreferences>Y</IntendedCompatibleWith
ClientsHavingSustainabilityPreferences>
  </ClientObjectives>
  <DistributionStrategy>
    <ExecutionOnly>B</ExecutionOnly>

<ExecutionWithCheckOrNonAdvisedServices>B</ExecutionWithCheckOrNonAdvisedServices>
  <InvestmentAdvice>B</InvestmentAdvice>
  <PortfolioManagement>B</PortfolioManagement>
  </DistributionStrategy>
</TargetMarket>
<CostsAndChargesExAnte>
  <Fund>
    <GrossMaxEntryCostNonAcquired>0.0500</GrossMaxEntryCostNonAcquired>
    <MaxExitCost>0.0000</MaxExitCost>
    <OngoingCosts>0.0175</OngoingCosts>
    <ManagementFee>0.0150</ManagementFee>
    <TransactionCosts>0.0012</TransactionCosts>
    <IncidentalCosts>0.0000</IncidentalCosts>
  </Fund>
  <CostsReferenceDate>2026-03-31</CostsReferenceDate>
</CostsAndChargesExAnte>
</FinancialInstrument>
</EMT_V4_2>

<!-- EFT: absent – this delivery flows from manufacturer to distributor,
      not distributor to manufacturer, so there is no feedback content. -->

<!-- EET: SFDR and sustainability data (mandatory minimum only) -->
<EET1.1.3>
  <EET1.1.3>
    <DataSetInformation>
      <DataSetInformation>
        <Version>V1.1.3</Version>
        <Producer>
          <Name>Europa Asset Management S.A.</Name>
          <LEI>529900T8BM49AURSD055</LEI>
          <Email>eet@europa-am.lu</Email>
        </Producer>
        <FileGeneration>2026-03-31T18:30:00Z</FileGeneration>
        <DataReporting>
          <SFDRPreContractual>Y</SFDRPreContractual>

```

```

        <SFDRPeriodic>Y</SFDRPeriodic>
        <SFDREntityLevel>Y</SFDREntityLevel>
        <MiFID>Y</MiFID>
        <IDD>Y</IDD>
    </DataReporting>
</DataSetInformation>
<ManufacturerInformation>
    <Name>Europa Asset Management S.A.</Name>
    <CodeType>L</CodeType>
    <Code>529900T8BM49AURSD055</Code>
    <Email>esg@europa-am.lu</Email>
    <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
</ManufacturerInformation>
    <ManufacturerWebsiteInformationStewardshipAndEngagement>https://www.europa-
am.lu/sustainability</ManufacturerWebsiteInformationStewardshipAndEngagement>
</DataSetInformation>
<GeneralFinancialInstrumentInformation>
    <IdentifyingData>LU2031234567</IdentifyingData>
    <TypeOfIdentification>1</TypeOfIdentification>
    <Name>Europa Growth Fund R EUR ACC</Name>
    <Currency>EUR</Currency>
</GeneralFinancialInstrumentInformation>
<MainCriterias>
    <SFDRProductType>8</SFDRProductType>
    <ConsidersPrincipleAdverseImpact>Y</ConsidersPrincipleAdverseImpact>
</MainCriterias>
    <MiFIDIDDTargetMarket>

<EndClientSustainabilityPreferences_Considered>Y</EndClientSustainabilityPreferences_Co
nsidered>
    </MiFIDIDDTargetMarket>
    <ScreeningCriteria>
        <ExistingNegativeScreeningStrategy>Y</ExistingNegativeScreeningStrategy>
    </ScreeningCriteria>
    <!-- Additional Article 8 sections (TaxonomyAlignedInvestments,
        PrincipalAdverseIndicators, etc.) are elided here and carried
        in full in Appendix D. -->
</EET1.1.3>
</EET1.1.3>

</DirectReporting>

<IndirectReporting>

    <!-- TPT: Solvency II lookthrough (abbreviated to one representative position) -->
    <TripartiteTemplateSolvencyII_V6>
        <Portfolio>
            <TPTVersion>V6.0</TPTVersion>
            <PortfolioID>
                <CodificationSystem>1</CodificationSystem>
                <Code>LU2031234567</Code>
            </PortfolioID>

```

```

<PortfolioName>Europa Growth Fund R EUR ACC</PortfolioName>
<PortfolioCurrency>EUR</PortfolioCurrency>
<TotalNetAssets>248500000.00</TotalNetAssets>
<ValuationDate>2026-03-31</ValuationDate>
<ReportingDate>2026-03-31</ReportingDate>
<CompleteSCRDelivery>Y</CompleteSCRDelivery>
<Positions>
  <Position>
    <InstrumentCIC>DE31</InstrumentCIC>
    <InstrumentCode>
      <CodificationSystem>1</CodificationSystem>
      <Code>DE0007164600</Code>
    </InstrumentCode>
    <InstrumentName>SAP SE</InstrumentName>
    <Valuation>
      <QuotationCurrency>EUR</QuotationCurrency>
      <MarketValueQC>7450000.00</MarketValueQC>
      <CleanValueQC>7450000.00</CleanValueQC>
      <MarketValuePC>7450000.00</MarketValuePC>
      <CleanValuePC>7450000.00</CleanValuePC>
      <PositionWeight>0.0300</PositionWeight>
      <MarketExposureQC>7450000.00</MarketExposureQC>
      <MarketExposurePC>7450000.00</MarketExposurePC>
      <MarketExposureWeight>0.0300</MarketExposureWeight>
    </Valuation>
    <AdditionalInformation/>
  </Position>
  <!-- Remaining 154 positions elided; full lookthrough in Appendix D. -->
</Positions>
</Portfolio>
</TripartiteTemplateSolvencyII_V6>

<!-- EPT (inside PRIIPS_V20): KID-input data for the PRIIPs disclosure -->
<PRIIPS_V20>
  <EPTV2>
    <DataSetInformation>
      <Version>V21</Version>
      <Data_ReportingNarratives>YES</Data_ReportingNarratives>
      <DataReportingCosts>YES</DataReportingCosts>

    <DataReportingAdditionalRequirementsGermanMOPs>NO</DataReportingAdditionalRequirementsGermanMOPs>

    <AdditionalInformationStructuredProductsRIY>NO</AdditionalInformationStructuredProductsRIY>
  </DataSetInformation>
  <GeneralPortfolioInformation>
    <PortfolioManufacturerName>Europa Asset Management S.A.
  </PortfolioManufacturerName>
    <PortfolioManufacturerGroupName>Europa Financial
  Group</PortfolioManufacturerGroupName>
    <PortfolioManufacturerLEI>529900T8BM49AURSD055</PortfolioManufacturerLEI>

```

```

    <PortfolioID>
      <CodificationSystem>1</CodificationSystem>
      <Code>LU2031234567</Code>
    </PortfolioID>
    <PortfolioName>Europa Growth Fund R EUR ACC</PortfolioName>
    <PortfolioCurrency>EUR</PortfolioCurrency>
    <GeneralReferenceDate>2026-03-31</GeneralReferenceDate>
    <PortfolioPRIIPSCategory>2</PortfolioPRIIPSCategory>
    <IsAnAutocallableProduct>N0</IsAnAutocallableProduct>
  </GeneralPortfolioInformation>
  <RiskAssessment>
    <ValuationFrequency>252</ValuationFrequency>
    <IS_Flexible>N0</IS_Flexible>
    <Existing_Credit_Risk>N0</Existing_Credit_Risk>
    <SRI>4</SRI>
    <IsSRIAdjusted>N0</IsSRIAdjusted>
    <MRM>4</MRM>
    <Recommended_Holding_Period>5</Recommended_Holding_Period>
    <HasAContractualMaturityDate>N0</HasAContractualMaturityDate>
    <Liquidity_Risk>L</Liquidity_Risk>
  </RiskAssessment>
  <PerformanceScenario>
    <ReturnUnfavorable>

    <RHP0rFirstCallDateOrFirstCallDate>-0.0120</RHP0rFirstCallDateOrFirstCallDate>
    </ReturnUnfavorable>
    <ReturnModerate>

    <RHP0rFirstCallDateOrFirstCallDate>0.0540</RHP0rFirstCallDateOrFirstCallDate>
    </ReturnModerate>
    <ReturnFavorable>

    <RHP0rFirstCallDateOrFirstCallDate>0.1080</RHP0rFirstCallDateOrFirstCallDate>
    </ReturnFavorable>
    <ReturnStress>

    <RHP0rFirstCallDateOrFirstCallDate>-0.2750</RHP0rFirstCallDateOrFirstCallDate>
    </ReturnStress>

    <PortfolioPastPerformanceDisclosureRequired>YES</PortfolioPastPerformanceDisclosureRequired>
  </PerformanceScenario>
  <Costs>
    <OneOff>
      <EntryCost>0.0500</EntryCost>
      <EntryCostsAcquired>0.0000</EntryCostsAcquired>
      <ExitCostAtRHP>0.0000</ExitCostAtRHP>
      <SlidingExitCostIndicator>N0</SlidingExitCostIndicator>
    </OneOff>
    <Ongoing>
      <OtherCost>0.0025</OtherCost>
      <ManagementCosts>0.0150</ManagementCosts>

```

```

        <Transaction>0.0012</Transaction>
    </Ongoing>
    <Incidental>
        <ExistringIncidentalCostsPortfolio>N0</ExistringIncidentalCostsPortfolio>
        <ExistingIncidentalCostsPortfolio>N0</ExistingIncidentalCostsPortfolio>
    </Incidental>
</Costs>
<Narratives>
    <ComprehensionAlert>N0</ComprehensionAlert>
    <IntendedTargetMarketRetailInvestor>Retail investors with basic or informed
knowledge of equity funds, willing to accept market risk, seeking long-term capital
growth over at least five years.</IntendedTargetMarketRetailInvestor>
    <InvestmentObjective>The fund aims to achieve long-term capital growth by
investing primarily in European large-cap equities that meet the manager's
sustainability criteria.</InvestmentObjective>
    <OtherMaterialyRelevantRiskNarrative>No other materially relevant risks
beyond market and sustainability risk.</OtherMaterialyRelevantRiskNarrative>
    <TypeUnderlyingInvestmentOption>UCITS SICAV</TypeUnderlyingInvestmentOption>
    <CapitalGuarantee>N0</CapitalGuarantee>
    <OneOffCostPortfolioExitCostDescription>No exit cost is charged at
redemption.</OneOffCostPortfolioExitCostDescription>
    <OngoingCostsPortfolioManagementCostsDescription>Ongoing management and
operating costs deducted from fund NAV.
</OngoingCostsPortfolioManagementCostsDescription>
    <DoCostsDependOnInvestedAmount>N0</DoCostsDependOnInvestedAmount>
</Narratives>
</EPTV2>
</PRIIPS_V20>

</IndirectReporting>

</RegulatoryReportings>

```

A three-pass reading of the block closes the example.

First pass — presence and absence. Four templates are present: EMT (MiFID II, under `DirectReporting`), EET (SFDR, also under `DirectReporting`), TPT (Solvency II, under `IndirectReporting`), and EPT inside `PRIIPS_V20` (PRIIPs, also under `IndirectReporting`). One template — EFT — is absent, and the absence is both visible (marked by a comment) and deliberate. A consumer who expects EFT in this delivery is confused about the delivery's direction; a consumer who correctly identifies the delivery as manufacturer-to-distributor accepts the absence as normal. The two-branch structure (`DirectReporting` versus `IndirectReporting`) is itself a structural feature that is worth highlighting: every FundsXML `RegulatoryReportings` block either places templates into these branches or is empty.

Second pass — shared identification across modules. Every one of the four present templates contains the same fund identifier (`LU2031234567`, the R-EUR-ACC share class) and the same manufacturer LEI (`529900T8BM49AURSD055`, Europa Asset Management S.A.). A consumer can read any single module and verify which fund and which manufacturer it refers to. The reference date (31 March 2026) appears under each module's own date field and is implicit in the enclosing `ContentDate` from the `ControlData` block (Chapter 4). The four modules are, in this sense, different views of the same product at the same moment in time.

Third pass — module-specific disclosures. Each module contributes its own distinctive content. EMT provides the target market classification, the forward-looking cost disclosure (1.75% ongoing + 0.12% transaction), and the distribution strategy. EET provides the SFDR Article 8 classification, the PAI consideration flag, the MiFID II sustainability preferences flag, and the existence of a negative screening strategy — all in their mandatory minimum form, with the full Article 8 disclosures (Taxonomy alignment, detailed PAI indicators) deferred to the unabbreviated version in Appendix D. TPT provides the fund's total NAV (248.5 million EUR), the 155-position lookthrough (abbreviated here to a single SAP SE position), and the CIC-coded asset classification that drives the insurer's SCR calculation. EPT, inside `PRIIPS_V20`, provides the SRI of 4, the four PRIIPs performance scenarios (from -27.5% annualised in stress to +10.8% favourable), the PRIIPs-specific cost breakdown, and the nine mandatory narrative fields that populate the KID's text sections. Each consumer reads the module relevant to its own purpose and ignores the others.

The complete block — with all 120+ EMT fields, the full PRIIPs v2.0 field set, all several hundred EET fields, and the full 155-position TPT lookthrough — appears in Appendix D as part of the end-to-end Europa Growth Fund sample file. This chapter's version is deliberately compressed to focus on the structural patterns.

8.10 Common Pitfalls

The following short list captures the mistakes that, in our experience, cause the greatest share of regulatory-module-related production incidents.

- **Templates placed at the wrong level of `RegulatoryReportings`.** A producer places `<EMT_V4_2>` directly inside `<RegulatoryReportings>` instead of inside `<DirectReporting>`, or places `<TripartiteTemplateSolvencyII_V6>` inside `<DirectReporting>` instead of `<IndirectReporting>`. The file fails schema validation. The rule is fixed by the XSD: EMT, EFTs, and EET belong in `DirectReporting`; `PRIIPS_V20` and `TripartiteTemplateSolvencyII_V6` belong in `IndirectReporting`; and neither branch may be skipped by lifting its children up one level.
- **Mixing `Y / N` with `YES / NO`.** EMT's `EMTYesNoType` uses single-letter `Y / N / Neutral`; EPT's (and the `PRIIPS_V20`'s) `YesNoType` uses `YES / NO` in full; EET's `EETYesNoType` uses `Y / N`. A producer that copies field values mechanically from one template to another will produce `Y` in a PRIIPs field where the schema expects `YES`, and the file fails validation. The fix is to use template-specific helpers when emitting the boolean fields and to test each template under xmllint.
- **EMT and EPT cost fields confused.** A consumer takes the annual total from EMT and treats it as the five-year-amortised figure that PRIIPs uses, or vice versa. The two figures are related (one-year costs × RHP, roughly) but they are not interchangeable, and the disclosure document generated will show the wrong figure. The rule: match the cost horizon to the disclosure horizon, and read the template appropriate to the disclosure being produced.
- **SFDR Article classification missing or wrong in EET.** A producer emits an EET block with `<SFDRProductType>` populated with the wrong digit — typically `6` (Article 6) on an Article 8 fund because an internal classification table had the wrong default. The consumer processes the fund as Article 6 and silently suppresses its ESG disclosures, misrepresenting the product. The field is mandatory and its value is business-critical; every EET must carry the correct value, and the producer's QA pipeline should cross-check against the fund's prospectus.

- **TPT lookthrough inconsistent with the native portfolio.** The `<Positions>` block inside TPT shows 155 positions, but the `<Portfolios>` block in the same delivery shows 153, because the TPT generator and the portfolio generator were run on different snapshots. An insurance consumer notices the discrepancy and raises a data quality flag. The fix is to produce both blocks from a single canonical source and to validate internal consistency before emitting.
- **Old template version in use.** A producer emits `<EMT_V4_1>` while its consumer expects `<EMT_V4_2>`. Both elements exist in the schema (FundsXML keeps every historical FinDatEx version), but a consumer's processing pipeline may only handle the current version, and the older element is silently ignored. The fix is to coordinate template version upgrades bilaterally and to migrate to the current version as soon as it is supported by all relevant consumers.
- **EFT included in a manufacturer delivery.** A producer (the asset manager) mistakenly includes an `<EFTs>` block in its own manufacturer-to-distributor delivery, perhaps because the producer's generator offers an "emit all templates" option and the operator forgot to deselect EFT. Consumers either reject the module as misplaced or log warnings. The fix is to enforce the direction rule in the producer's tooling: manufacturers produce EMT/EPT/EET/TPT; distributors produce EFT; neither produces the other's content.
- **Minimal EET emitted for an Article 8 fund.** A producer emits only the mandatory minimum of the EET — the skeleton shown in §8.9 — for an Article 8 fund that should carry a substantial additional payload of Taxonomy alignment and PAI data. The file passes schema validation but fails downstream semantic checks at the distributor, which expects the richer Article 8 content and flags the delivery as semantically incomplete. The fix is to populate the optional Article 8 / Article 9 sections whenever the fund is classified accordingly, even though the schema does not force the content.

8.11 Key Takeaways

- `RegulatoryReportings` is the fifth and final main area of a FundsXML delivery, a root-level sibling of `<Funds>` and `<AssetMasterData>`. Inside, it immediately splits into `<DirectReporting>` and `<IndirectReporting>`, and the five FinDatEx templates are distributed between the two branches.
- FinDatEx maintains the templates; FundsXML embeds them. The two organisations have clearly separated responsibilities — FinDatEx decides the *what*, FundsXML decides the *how*. Each FinDatEx template version appears in the FundsXML schema as its own versioned element (e.g. `EMT_V4_2`, `EET1.1.3`, `PRIIPS_V20`, `TripartiteTemplateSolvencyII_V6`).
- **EMT** (under `DirectReporting`, as `EMT_V4_2`) carries MiFID II cost and target market data at the share-class level and is the operationally most important template for retail distribution. Its `FinancialInstrument` complex type has five mandatory sections: `DataSetInformation`, `GeneralInformation`, `TargetMarket`, `CostsAndChargesExAnte`, and the optional `CostsAndChargesExPost`.
- **EPT** (under `IndirectReporting`, as `PRIIPS_V20/EPTV2`) provides the data that populates the PRIIPs KID: the SRI on the 1-to-7 scale, the four performance scenarios as annualised returns over the recommended holding period, the PRIIPs-specific cost breakdown, and the mandatory narrative fields. Several of the boolean fields use `YES / NO` in full, unlike EMT's single-letter flags.

- **EET** (under `DirectReporting`, as `EET1.1.3`) covers SFDR classification (Articles 6, 8, 9), the fourteen mandatory Principal Adverse Impacts, and the EU Taxonomy alignment. It is by far the largest template by field count, with several hundred mostly optional fields organised into sections for main criteria, MiFID/IDD target market, screening, Taxonomy alignment, and PAI. The mandatory minimum is compact, but Article 8 and Article 9 funds are expected to populate the optional sections substantively.
- **EFT** (under `DirectReporting`, as `EFTs`) flows in the reverse direction (distributor to manufacturer) and therefore does not appear in manufacturer deliveries like the Europa Growth Fund's monthly file. EFT carries distribution statistics, reference target market specification, and deviation reports under the manufacturer or distributor target market perspective.
- **TPT** (under `IndirectReporting`, as `TripartiteTemplateSolvencyII_V6`) provides lookthrough data for Solvency II SCR calculations by insurance investors. Each `Portfolio` entry carries mandatory top-level fields (`TPTVersion=V6.0`, `PortfolioID`, `PortfolioName`, `PortfolioCurrency`, `TotalNetAssets`, `ValuationDate`, `ReportingDate`, `CompleteSCRDelivery`) plus one or more `Position` entries, each with an `InstrumentCIC` classification and six mandatory valuation sub-fields.
- **ESAP** is the EU-wide access point for regulatory disclosures, live in phased rollout from 2026. FundsXML 4.2.8 has no schema-level ESAP metadata element; ESAP routing is handled by the production pipeline outside the schema, and producers publish SFDR disclosures, prospectuses, and similar public documents through a separate ESMA-specified channel.
- The Europa Growth Fund's 31 March 2026 delivery carries EMT, EPT, EET, and TPT — but not EFT — reflecting the direction of the delivery and the reporting needs of its consumers.

We have now covered all five main areas of a FundsXML delivery — `ControlData`, `Funds`, `AssetMasterData`, the four substructures of `FundDynamicData`, and `RegulatoryReportings`. What remains are the less-frequently-used but operationally important schema areas: the `Documents` section for fact-sheets and signed attachments, XML digital signatures, `CustomAttributes` for proprietary extensions, and country-specific additions. Chapter 9 closes Part II of the book with these advanced topics.



PART II — FUNDSXML IN DETAIL · CHAPTER 9

Advanced Schema Areas

Factsheets, digital signatures, and CustomDataFields

9.1 Setting the Scene: The Rest of the Schema

At the end of Chapter 8 we had covered all five main areas of a FundsXML delivery — `ControlData`, `Funds`, `AssetMasterData`, the four substructures of `FundDynamicData`, and `RegulatoryReportings`. Those five areas carry the overwhelming majority of what every production FundsXML document transports, and they are the subject of every chapter of Part II up to this point. What remains are four quieter but operationally important schema areas: the `Documents` section for factsheets, KIDs, and other attachments; `ds:Signature`, the XML digital-signature element the schema imports from the W3C XMLDSig specification; the `CustomAttributes` extension mechanism; and the `CountrySpecificData` branches that carry national regulatory extensions. This chapter treats all four, and it does so against the real schema.

A disclosure about this chapter's code listings is worth making up front, because it changes the rule that Chapters 4 through 8 followed. Every XML example in Chapters 4 to 8 was written for pedagogical clarity — element names, field orderings, and occasional attribute values were shaped around the narrative rather than against the current FundsXML 4.2.8 XSD. The corrections applied retroactively to those chapters fix the most egregious deviations (the `DataOperation` enumeration, the `CustomAttributes` name, the namespace declaration, and a few other structural issues), but the deeper element-by-element alignment with the production schema was not attempted. Chapter 9 changes that rule: **every XML listing in this chapter has been validated against `FundsXML4.xsd` using `xmllint`**, and the validation command is the same for every example:

```
xmllint --noout --schema FundsXML4.xsd <example>.xml
```

Readers who want to reproduce the validation can copy any listing into a file, run the command, and expect to see `<example>.xml validates` on standard output. The listings carry the full document root, including the namespace-less `FundsXML4` element with `xsi:noNamespaceSchemaLocation`, so that each one is a self-contained, schema-valid file rather than a fragment.

By the end of this chapter, you should be able to:

- populate the `Documents` section with factsheets, PRIIPs KIDs, and annual reports, choosing correctly between the `DocumentURL` and `BinaryData` content modes;
- add an enveloped XML digital signature (XMLDSig) to a FundsXML delivery and explain its relationship to the separate `Signature` flag on a `Document` entry;

- use `CustomAttributes` to carry proprietary extension data without breaking the schema, and recognise when a `CustomAttributes` entry is the right tool and when a schema change request is the right tool;
- populate the country-specific branches in both `ControlData` and at the root of the document for Austria, Luxembourg, and the other supported jurisdictions;
- read a FundsXML document whose root element contains all four advanced areas and explain which consumers read which parts.

The chapter proceeds from the biggest of the four topics to the smallest: Documents in §9.2, XML digital signatures in §9.3, `CustomAttributes` in §9.4, country-specific additions in §9.5. A short integrative section (§9.6) shows how the four advanced areas fit alongside the five main areas of Chapters 4 to 8, and the chapter closes with the usual pitfalls and takeaways.

9.2 The Documents Section — Factsheets, KIDs, Signed Attachments

The `Documents` element is a root-level sibling of `Funds` and `AssetMasterData` in a FundsXML document. It carries references to — or, in some deployments, the actual binary content of — fund-related documents that travel alongside the structured data: factsheets, prospectuses, key information documents, annual reports, audit reports, and anything else the asset manager wants to deliver as a formal document rather than as structured data. Chapter 3 introduced the `Documents` section as one of the five main areas of a FundsXML delivery; this section treats it in detail.

9.2.1 Why a Documents Section Exists at All

The first question a reader might reasonably ask is why FundsXML, which is already a structured-data format, bothers with a mechanism for carrying *documents* at all. The answer is that structured data and documents are not substitutes for each other. A PRIIPs KID is a regulatory artefact that has to be delivered to investors *as a formal document* — typically a PDF — regardless of whether the underlying data can also be delivered in structured form. The regulator does not accept the EPT (Chapter 8) as a replacement for the KID; the EPT is an input to the KID-generation process, but the KID itself is a PDF that must be delivered separately and that every retail investor has the right to see before they buy.

The same argument applies to prospectuses, annual reports, audit reports, and AIFMD filings. All of these are documents with formal legal status, and the FundsXML `Documents` section exists so that a single delivery can carry both the structured data *and* the formal documents that reference that data, without forcing the producer and the consumer to maintain two parallel delivery channels.

9.2.2 The Structure of a Document Entry

A `Documents` element contains one or more `Document` children. Each `Document` carries a set of fields that describe the document's type, language, applicability, and technical format, plus one of two alternative content representations — a URL pointing at the actual file, or the file's contents embedded directly inside the XML as base64-encoded binary data.

The main fields are:

- **Type** — a choice between a **ListedType** (an enumeration with the values *AIFMD*, *AnnualReport*, *AuditReport*, *Factsheet*, *KID*, *Prospectus*, and *PRIIPS-KID*) and an **UnlistedType** (a free-text string for document kinds not covered by the enumeration). Every **Document** must pick exactly one of the two.
- **Version** — an optional version string, typically a date (*2026-03-31*) or a release identifier (*2025*).
- **Language** — an `xs:language` code identifying the language of the document's content.
- **TranslationsAvailable** — a Boolean flag indicating whether the same document is available in other languages.
- **PublicationCountries** — a list of ISO country codes naming the countries for which the document is valid.
- **Fund** , **Subfunds** , **ShareClasses** — optional backward pointers to the fund, sub-funds, or share classes the document applies to. A PRIIPs KID is per-share-class and therefore references a specific **ShareClass** ; an annual report is typically fund-level and references the **Fund** .
- **Name** — a human-readable name for the document.
- **FileName** — the name of the file (including extension), useful when a consumer needs to save the document to disk.
- **ForPublicUsage** — a Boolean indicating whether the document may be redistributed to the public.
- **Format** — the technical format: *PDF*, *XML*, *Excel*, *Word*, *Image*, and so on.
- **Signature** — an enumeration (*No*, *Digital*, *Scan*) indicating whether the document is signed, and if so by what method. This is a *flag*, not a signature itself; the cryptographic signature, if any, is carried separately (see §9.3).
- **CreationDate** , **ModificationDate** , **ExpirationDate** — the timestamps that govern the document's lifecycle.
- **DataSupplier** — the party that produced the document.
- **SizeInBytes** — the document's size (useful for consumers that need to budget download quotas).
- **DocumentURL** or **BinaryData** — the actual content.

9.2.3 URL versus Embedded Content

The choice between **DocumentURL** and **BinaryData** is the one operational decision a producer makes for every document. Both are legitimate in FundsXML, and each has clear advantages.

DocumentURL contains an HTTPS (or occasionally SFTP) URL pointing at a location where the consumer can download the document on demand. The advantages are: the FundsXML file itself stays small, regardless of how many multi-megabyte PDFs the delivery references; the document can be updated after the FundsXML file has been shipped, as long as the URL still resolves; and the producer retains control over who can access the document by managing the HTTPS endpoint's authentication. The disadvantage is that the FundsXML file and the documents it references are *not* self-contained — a consumer that archives the FundsXML file but fails to download the referenced documents at the same time may find later that the URLs have expired or been moved.

BinaryData embeds the document's bytes directly into the FundsXML file as base64-encoded content. The advantages are: perfect self-containedness (the consumer gets everything in one file); no dependency on any external HTTPS infrastructure; and the possibility of cryptographically binding the document to the FundsXML envelope

through the enveloped signature mechanism of §9.3. The disadvantages are the file-size explosion (a 2-megabyte PDF becomes a 2.7-megabyte base64 blob inside the XML) and the rigidity (once embedded, the document cannot be updated without re-issuing the whole FundsXML delivery).

The Europa Growth Fund's production convention, which Chapter 13 will formalise as a project decision, is **DocumentURL for the fund's routine monthly deliveries** (where bandwidth matters and the documents rarely change between deliveries) and **BinaryData for the quarterly or annual ESAP submissions** (where self-containedness is a regulator requirement). The example in §9.2.4 shows **DocumentURL** because that is the more common case.

9.2.4 A Validated Example: Two Documents for the Europa Growth Fund

The complete FundsXML document below carries two entries in its **Documents** section: the PRIIPs KID for the R-EUR-ACC share class, and the fund's annual report for the 2025 financial year. Both entries use **DocumentURL** for the content. The full document validates against the current FundsXML 4.2.8 schema with `xmllint --noout --schema FundsXML4.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-DOC-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
    <DataOperation>INITIAL</DataOperation>
    <Language>en</Language>
  </ControlData>
  <Documents>
    <Document>
      <Type>
        <ListedType>PRIIPS-KID</ListedType>
      </Type>
      <Version>2026-03-31</Version>
      <Language>en</Language>
      <TranslationsAvailable>true</TranslationsAvailable>
      <PublicationCountries>
        <Country>LU</Country>
        <Country>DE</Country>
        <Country>FR</Country>
      </PublicationCountries>
      <ShareClasses>
        <ShareClass>
```

```

        <Identifiers>
            <ISIN>LU2100000011</ISIN>
        </Identifiers>
        <Name>Europa Growth Fund R EUR ACC</Name>
        <Currency>EUR</Currency>
    </ShareClass>
</ShareClasses>
<Name>Europa Growth Fund R EUR ACC – PRIIPs KID (English)</Name>
<FileName>EGF_R_EUR_ACC_KID_en_20260331.pdf</FileName>
<ForPublicUsage>true</ForPublicUsage>
<Format>PDF</Format>
<Signature>Digital</Signature>
<CreationDate>2026-03-31</CreationDate>
<ExpirationDate>2027-03-31</ExpirationDate>
<DataSupplier>
    <SystemCountry>LU</SystemCountry>
    <Short>EAM</Short>
    <Name>Europa Asset Management S.A.</Name>
    <Type>IC</Type>
</DataSupplier>
<DocumentURL>https://docs.europa-asset-
management.com/kids/LU2100000011_en_20260331.pdf</DocumentURL>
</Document>
<Document>
    <Type>
        <ListedType>AnnualReport</ListedType>
    </Type>
    <Version>2025</Version>
    <Language>en</Language>
    <Fund>
        <Identifiers>
            <LEI>549300ABCDEFGHIIJ1234</LEI>
        </Identifiers>
        <Name>Europa Growth Fund</Name>
        <Currency>EUR</Currency>
    </Fund>
    <Name>Europa Growth Fund – Annual Report 2025</Name>
    <FileName>EGF_AnnualReport_2025.pdf</FileName>
    <ForPublicUsage>true</ForPublicUsage>
    <Format>PDF</Format>
    <Signature>Digital</Signature>
    <CreationDate>2026-03-15</CreationDate>
    <DocumentURL>https://docs.europa-asset-
management.com/annual/EGF_2025.pdf</DocumentURL>
    </Document>
</Documents>
</FundsXML4>

```

Reading the block in two passes.

The **first pass** looks at the PRIIPs KID entry. Its type is identified by the `ListedType` enumeration value `PRIIPS-KID`, one of the seven values the schema accepts. The version `2026-03-31` tags this as the version of the KID that was valid as of the quarter-end delivery date. The language is English; `TranslationsAvailable=true` tells a consumer that this KID exists in other languages too and can be requested separately. The KID applies to three distribution countries (Luxembourg, Germany, France — the countries where the R-EUR-ACC class is distributed to retail investors at the time of the delivery). The backward pointer to the share class uses the `ISIN` of R-EUR-ACC, matching the value introduced in Chapter 5. The file is a PDF, it is signed digitally, it was created at the valuation date, and it expires one year later — the PRIIPs regulatory cycle. The data supplier block is repeated inside the `Document` element, letting a consumer identify the producer of the document independently of the enclosing file's producer (in this case they are the same, but they need not be). Finally, `DocumentURL` gives the HTTPS location where the document can be retrieved on demand.

The **second pass** looks at the annual report entry. Its type is `AnnualReport`, another of the seven `ListedType` values. It is associated with the *fund as a whole* rather than with an individual share class, so the backward pointer uses the fund's `Fund/Identifiers/LEI` rather than a share-class ISIN. The `Language` is English, but no `PublicationCountries` list is given because an annual report is a fund-level document that is not restricted to particular distribution countries. No `ExpirationDate` is set because an annual report does not expire in the regulatory sense; it stays valid until superseded by the next year's report.

Both entries have `Signature=Digital`, indicating that the underlying PDFs are digitally signed. The actual cryptographic signatures, however, are on the PDFs themselves, not inside the FundsXML file. The next section treats the separate case of signing the entire FundsXML document.

9.3 XML Digital Signatures

The root element of a FundsXML document, after the five main content areas we have already covered and the optional `CountrySpecificData` branch that §9.5 treats, allows an optional `ds:Signature` element as its final child. This is the XML digital-signature element defined by the W3C XMLDSig specification, imported into the FundsXML schema through the `xmlsig-core-schema.xsd` file that sits alongside `FundsXML4.xsd`. When present, it signs the containing FundsXML document cryptographically, so that any downstream consumer can verify — with the usual public-key-infrastructure machinery — that the document was produced by the claimed signer and has not been tampered with since signing.

9.3.1 Why Sign FundsXML

The case for signing FundsXML documents rests on three distinct needs.

Integrity is the most obvious. A FundsXML file that passes through multiple hands between its producer and its final consumer — a fund administrator, a transfer mechanism, a distributor's staging system, a downstream analytics pipeline — can be subject to deliberate or accidental modification at any of those stages. A cryptographic signature binds the document's content to its producer's identity in a way that a verifier can check independently: if any byte of the signed content has changed since the signature was created, verification fails. Integrity matters particularly for regulatory submissions and for data that may later be used as evidence in a dispute.

Non-repudiation is the second case. In some bilateral relationships — particularly where fund data is used as the basis for financial settlement or regulatory filing — the consumer needs to be able to prove that a particular document came from a particular producer, *and* that the producer cannot later deny having sent it. A valid digital signature gives exactly this property: the signer, having used their private key to sign the document, cannot credibly claim that somebody else signed it in their name (provided the private key was properly protected, which is a key-management concern rather than a schema concern).

Authentication, the third case, overlaps with the first two but is worth naming separately. A consumer that receives a signed FundsXML document can determine the signer's identity from the certificate embedded in the signature — typically an `X509Certificate` or `X509SubjectName` field inside the `KeyInfo` block — and match that identity against an allowlist of approved producers. This is a stronger form of the sender-matching we discussed in Chapter 4 (where the allowlist matched on the `DataSupplier/LEI` text in the `ControlData` block); a signature-based allowlist cannot be forged simply by writing the right LEI into the XML, because the private key needed to produce a valid signature is held only by the real producer.

9.3.2 Enveloped, Enveloping, and Detached Signatures

XMLDSig defines three signature forms, and the choice between them matters for where the signature lives and what it signs.

In an **enveloped** signature, the `ds:Signature` element sits *inside* the document being signed, typically as its last child. The signature covers the enclosing document minus the signature element itself (the `enveloped-signature` transform excludes the signature from the content it signs, to avoid a circular dependency). This is the form FundsXML uses natively, because the FundsXML schema explicitly allows `ds:Signature` as the last optional child of the root `FundsXML4` element.

In an **enveloping** signature, the reverse applies: the signed content sits inside a `ds:Object` element *within* the signature. The top-level element is the signature itself. This form is useful when a signature needs to bundle arbitrary content together with its own metadata, but it does not fit FundsXML's document-centric design.

In a **detached** signature, the signature and the content are separate documents. The `ds:Reference` element inside `ds:SignedInfo` uses a URI to point at the content. Detached signatures are useful when the signing party cannot or does not want to modify the signed document at all — for example, when signing a PDF that has already been published and must remain bit-for-bit identical. FundsXML does not typically use detached signatures at the envelope level, but the individual `Document` entries in the `Documents` section frequently reference PDFs whose own signatures are detached (the `Signature=Digital` flag we saw in §9.2 indicates this case).

For the rest of this section we treat the enveloped form, which is the one natively supported by FundsXML at the root.

9.3.3 Canonicalization and Digest Algorithms

Cryptographic signatures operate on bytes, not on XML. When a consumer verifies a signature, it computes a hash of the signed content and compares it with the digest value in the signature. But XML has many equivalent byte-level representations of the same logical content — different whitespace, different attribute orderings, different namespace prefixes, different quote characters — and a naive hash would fail verification whenever the document passed through an intermediate parser that re-serialised it in a different form.

The XMLDSig standard solves this problem through **canonicalisation**: a well-defined procedure that converts any XML into a unique canonical byte form, such that two logically equivalent XML documents always produce the same bytes. The canonical form is the thing that actually gets hashed.

The FundsXML examples in this section use **Canonical XML 1.0** (<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>) as the canonicalisation algorithm and **SHA-256** (<http://www.w3.org/2001/04/xmllenc#sha256>) as the digest algorithm. The signature itself uses **RSA-SHA256** (<http://www.w3.org/2001/04/xmldsig-more#rsa-sha256>). These are the current best-practice defaults in 2026; older FundsXML deliveries may use SHA-1 and RSA-SHA1, which are now deprecated but still legal under the schema.

9.3.4 A Validated Example: A Signed FundsXML Document

The listing below is a FundsXML document with a minimal `ControlData` block and an enveloped `ds:Signature` at the root. The signature structure is complete and schema-valid: it has a `SignedInfo` with canonicalisation, signature, and digest methods, a single `Reference` to the document root (`URI=""` means the whole document), the transforms that strip the signature itself from the signed content, and a `SignatureValue` block. The `SignatureValue` in this example is a placeholder — a real signature value is produced by running an actual signing key over the canonicalised content, not by hand-writing a base64 string. Consumers that validate signatures *cryptographically* (as opposed to just *schematically*) will reject the placeholder; consumers that only care about schema validation (as we do in this chapter) will accept it.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-SIG-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
    <DataOperation>INITIAL</DataOperation>
    <Language>en</Language>
  </ControlData>
  <ds:Signature Id="EGF-DocumentSignature">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
      <ds:Reference URI="">
        <ds:Transforms>
```

```

        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
        <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig#sha256"/>
    <ds:DigestValue>9Fxs0VvC6QyzGx2rSc0oYQ3fwBp9tEjU5aKQUAJJSWM=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
    Krl3V4o9E8mQy5q3FqJdqXm0r4qYvHhj3HwNpfqX7l+yXYQeJqmJ4t6VvZa0MwvFH+
    uk10JmI7mYn3Wx2PcNpRaVFqEGxCDzYbLdmCqN6mTgVrJLmhd3QoJzTkYvNbXeZBqF
    LJK6y9V3ZX2mPvnRJu6C8XhQwdmFjW4Vx9M5P1XyJcHL3FtV1pnKZBXsh8fQKL2rL4
    PcZMnL5x3WfHp0Yh2mN+tvQJkRBk1xC3D6oCqmYwHbKv09pF8yA7jKL3vYQ5nNq3Yx
    fR2zHLnVqKbPjWcMpI3y0nM2oNpE4L3+kMwZg7HVQKrDzLJ0YJ9tPrX5wVpJ7hKy8Q
    ==
</ds:SignatureValue>
<ds:KeyInfo>
    <ds:X509Data>
        <ds:X509SubjectName>CN=Europa Asset Management S.A., O=Europa Asset Management,
C=LU</ds:X509SubjectName>
    </ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
</FundsXML4>

```

Reading the signature block in three passes.

The **first pass** looks at `SignedInfo`. It carries three critical algorithm declarations: the canonicalisation method (Canonical XML 1.0), the signature method (RSA with SHA-256), and the reference to the signed content. The reference URI is the empty string, which in XMLDSig convention means "the document containing this signature". The two transforms strip the signature itself from the content (so that the signature does not sign itself) and then canonicalise what remains.

The **second pass** looks at the digest and the signature value. `DigestValue` contains the base64-encoded SHA-256 hash of the canonicalised, enveloped-signature-stripped FundsXML document. A verifier computes the same hash from its local copy of the document and compares. `SignatureValue` contains the RSA-SHA256 signature of the `SignedInfo` block, computed with the producer's private key. A verifier uses the public key (from the `KeyInfo` block, or from a separately distributed certificate) to decrypt the signature and compare against its own hash of `SignedInfo`.

The **third pass** looks at `KeyInfo`. It carries an `X509SubjectName` identifying the signer — Europa Asset Management S.A., with its organisation and country. A full production signature typically includes an `X509Certificate` element with the base64-encoded certificate bytes, so that the verifier can check not only the signer's identity but also the certificate chain that vouches for it; we have omitted that for brevity. Consumers that match against an allowlist of approved signers look up the `X509SubjectName` (or the full certificate's fingerprint) and accept or reject accordingly.

9.3.5 The Signature Flag on Document Entries versus the Root Signature

The `Document` element in §9.2 carries a `Signature` field whose value is one of *No*, *Digital*, or *Scan*. This is **not** the same thing as the `ds:Signature` element we just discussed. The `Document/Signature` field is a *flag*: it tells the consumer whether the PDF (or other format) that the document references is itself signed, and if so whether the signature is cryptographic (*Digital*) or a scan of a hand-written signature (*Scan*). The cryptographic signature, if any, lives *inside the referenced document*, not inside the FundsXML file — typically as a PKCS#7 signature embedded in the PDF's own signature dictionary.

The root-level `ds:Signature` element, by contrast, signs *the FundsXML envelope* itself — everything inside the root `FundsXML4` element except the signature sub-tree. The two signature mechanisms are orthogonal: a FundsXML document can be signed at the root, it can contain signed PDFs (flagged by `Document/Signature=Digital`), it can do both, or it can do neither. Choosing which combination to use is a project-level decision that Chapter 13 will treat in the context of a full implementation project.

9.4 CustomAttributes — The Extension Mechanism

The final bespoke feature of FundsXML that we treat in this chapter is the schema's extension mechanism: a way for producers to attach arbitrary named key-value data to many of the standard element types without breaking the schema. The mechanism is called `CustomAttributes`, not `CustomDataFields` (an older and imprecise name that appeared in marketing material and that earlier chapters of this book have now been corrected to match the XSD).

9.4.1 Why an Extension Mechanism Exists

No standard can enumerate every field that every member firm, in every jurisdiction, will ever need. The FundsXML designers accepted this from the start and built an extensibility mechanism into the schema, so that producers with legitimate proprietary data could carry it through the standard without either forking the XSD or losing the data.

The extension mechanism is **structured**, not free-form. A producer cannot simply paste arbitrary XML into a reserved "extensions" region; the schema requires that every extension value be named, typed, and placed into a `CustomAttributes` element whose own structure is fixed. This deliberate constraint gives consumers a fighting chance of making sense of unknown fields: even if a consumer does not recognise a particular attribute name, it can still parse the attribute's name, type, and value according to the common structure, and decide individually whether to store, log, or ignore each one.

9.4.2 The Structure of a CustomAttributes Element

Every `CustomAttributes` element contains one or more `Attribute` children. Each `Attribute` has:

- **Name** — a `Text256Type` string identifying the attribute. Producers are encouraged to use a namespace-like convention (`eam.hedge.variantOf` , `distributor.channel.code`) to avoid name collisions.
- **Type** — a single-character enumeration with the values `T` (Text), `N` (Number), `D` (Date), or `B` (Boolean).

- **One of four value elements**, chosen to match the `Type`: `ValueText` for `T`, `ValueNumber` for `N`, `ValueDate` for `D`, `ValueBoolean` for `B`.

An `Attribute` that declares `Type=N` and then carries a `ValueText` child, or that declares `Type=T` and then carries a `ValueNumber`, is schema-invalid in spirit even though the XSD allows the mismatch through its `xs:choice` structure. Producers should always match the type flag to the value element.

The `CustomAttributes` element itself appears at many points in the FundsXML schema — as an optional last child of `ControlDataType`, `AssetType`, `FundStaticDataType`, and several other complex types. Consumers read it whenever they encounter it and treat the values according to their own knowledge of the attribute names. An unknown attribute is not an error; it is simply not interpreted.

9.4.3 A Validated Example

The listing below is a minimal FundsXML document whose `ControlData` block ends with a `CustomAttributes` element containing four entries: a text attribute identifying the producer's pipeline version, a numeric attribute naming the delivery's serial number within the month, a date attribute recording when the first-cut version of the delivery was produced, and a Boolean flag indicating whether the delivery is a replacement for an earlier one. The full document validates against the current schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-CA-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
    <DataOperation>INITIAL</DataOperation>
    <Language>en</Language>
    <CustomAttributes>
      <Attribute>
        <Name>eam.pipeline.generator</Name>
        <Type>T</Type>
        <ValueText>egf-batch v7.3.1</ValueText>
      </Attribute>
      <Attribute>
        <Name>eam.delivery.serial</Name>
        <Type>N</Type>
        <ValueNumber>472</ValueNumber>
      </Attribute>
      <Attribute>
        <Name>eam.delivery.firstCutSent</Name>
```

```

        <Type>D</Type>
        <ValueDate>2026-04-01</ValueDate>
    </Attribute>
    <Attribute>
        <Name>eam.amend.isReplacement</Name>
        <Type>B</Type>
        <ValueBoolean>>false</ValueBoolean>
    </Attribute>
</CustomAttributes>
</ControlData>
</FundsXML4>

```

The four attributes above have no meaning to any generic FundsXML consumer, and that is the point. A consumer that recognises the `eam.*` prefix — presumably because it has an integration contract with Europa Asset Management S.A. — reads the values and uses them for whatever purpose the contract defines. A consumer that does not recognise the prefix ignores the attributes entirely, parses the rest of the FundsXML document normally, and continues. No conformance is broken.

9.4.4 Governance and Best Practices

The power of `CustomAttributes` is also its danger. Every attribute a producer adds is a local extension that only a consumer who knows about it can use, and every such extension makes the FundsXML dialect of that producer slightly different from the global standard. Four governance rules minimise the cost of extensions over time.

Rule 1: Name attributes in a namespace-like convention. A producer that uses bare names like `status` and `batchNo` invites collision with any other producer that uses the same names, and consumers that merge data from multiple producers will not know which producer a given attribute came from. Names like `eam.delivery.status` and `bnp.batch.number` make the producer visible in the attribute name itself.

Rule 2: Use `CustomAttributes` for producer-specific operational metadata, not for fund content that has a real home in the schema. A producer who writes the fund's management fee into a `CustomAttributes/Attribute` because the pipeline was written before the producer understood the schema's native fee fields is causing trouble for every consumer. Content that belongs in a first-class schema field should live there; `CustomAttributes` is for the long tail of operational, pipeline-level, and truly proprietary data that the schema does not cover.

Rule 3: Prefer a schema change request over a permanent `CustomAttributes` entry. If a producer finds itself shipping the same `CustomAttributes` entry on every delivery for several months, the need is probably not truly proprietary — it is a gap in the standard, and the right response is to propose the field to the FundsXML working groups (Chapter 3.3.2) so that a future schema release can absorb it as a first-class element. Long-lived `CustomAttributes` are a smell that the standard needs to evolve.

Rule 4: Document every attribute the producer emits in a bilateral contract with each consumer that uses it. Without documentation, a consumer receiving an attribute it does not know cannot reliably decide whether to store it, warn about it, or ignore it. A short specification document accompanying the FundsXML delivery stream — typically updated whenever the producer adds or retires an attribute — keeps consumers aligned without requiring them to reverse-engineer the producer's schema.

9.5 Country-Specific Additions

The last of the four advanced schema areas is `CountrySpecificData`. This is the mechanism by which FundsXML carries national regulatory extensions that apply only in specific jurisdictions — the Austrian OeKB master data sheet, the Luxembourg CSSF fund-registration identifiers, the French AMF reporting extensions, and so on. The mechanism appears in two structurally distinct places in the schema, and understanding both is essential to reading a real production document.

9.5.1 Two Places, Two Purposes

The first place is **inside `ControlData`**. The `ControlData/CountrySpecificData` element carries country-specific *control-layer* data — metadata about the delivery itself that is specific to a particular national regulator. The Austrian version, for example, carries fields that the Oesterreichische Kontrollbank (OeKB) fund data portal needs to accept or reject a delivery; the other national sub-branches typically carry their own national equivalents or, where no national control metadata is defined, appear as free-form containers.

The second place is **at the root of the FundsXML document**, as a top-level `CountrySpecificData` element that sits alongside `Funds`, `AssetMasterData`, `Documents`, and `RegulatoryReportings`. The root-level `CountrySpecificData` carries country-specific *content* — national regulatory fields that are substantive rather than procedural. For Germany, this region is typed to carry PRIIPS-DE-EPT-Pia-specific characteristics (a national extension to the EPT regulatory template from Chapter 8). For other countries, the region is declared as `xs:anyType`, which gives producers the freedom to carry any XML content their national regulator requires, at the cost that consumers cannot validate the content without additional out-of-band schema knowledge.

Both places coexist in the same document: a FundsXML delivery can populate `ControlData/CountrySpecificData/AT` (for Austrian control-layer metadata) and `/CountrySpecificData/LU` (for Luxembourg content-layer extensions) in the same file, and consumers read each region independently.

9.5.2 The Country-Specific Module Architecture

The schema's `ModuleUsage` element inside `ControlData` (which we met in Chapter 4 and revisit here) lets a producer declare which country-specific sub-modules a delivery uses. Values include `CountrySpecificData_AT`, `CountrySpecificData_DE`, `CountrySpecificData_DK`, `CountrySpecificData_FR`, `CountrySpecificData_LU`, and `CountrySpecificData_NL`. A consumer that supports only a subset of these modules can read the `ModuleUsage` declaration and decide quickly whether the delivery is within its supported range, without having to parse the actual country-specific content first.

The modules themselves are defined in separate XSD files — `FundsXML4_CountrySpecificData_AT.xsd`, `FundsXML4_CountrySpecificData_LU.xsd`, and so on — that the main schema includes. In the project-local copy of `FundsXML4.xsd` used throughout this book, most country modules are declared with `xs:anyType`, giving producers maximum flexibility; the extended schemas in the full FundsXML distribution contain richer national-regulator-specific types that some producers and consumers use instead.

9.5.3 A Validated Example

The listing below shows a FundsXML document with `CountrySpecificData` populated in both places. The `ControlData` block carries an Austrian control-layer extension naming the OeKB fund data portal as the target registry, with two free-form information elements. The root-level `CountrySpecificData` carries a Luxembourg content extension with a fictional CSSF registration number and registration date for the fund. Both branches validate against the current schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-CS-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
    <DataOperation>INITIAL</DataOperation>
    <Language>en</Language>
    <CountrySpecificData>
      <AT>
        <FundDataPortalContent>OeKB Stammdatenblatt</FundDataPortalContent>
        <FundDataPortalInfos>
          <Info type="source">EAM-Luxembourg delivery gateway</Info>
          <Info type="note">Monthly month-end reporting</Info>
        </FundDataPortalInfos>
      </AT>
    </CountrySpecificData>
  </ControlData>
  <CountrySpecificData>
    <LU>
      <CSSFRegistration>
        <RegistrationNumber>000001234</RegistrationNumber>
        <RegistrationDate>2012-01-15</RegistrationDate>
      </CSSFRegistration>
    </LU>
  </CountrySpecificData>
</FundsXML4>
```

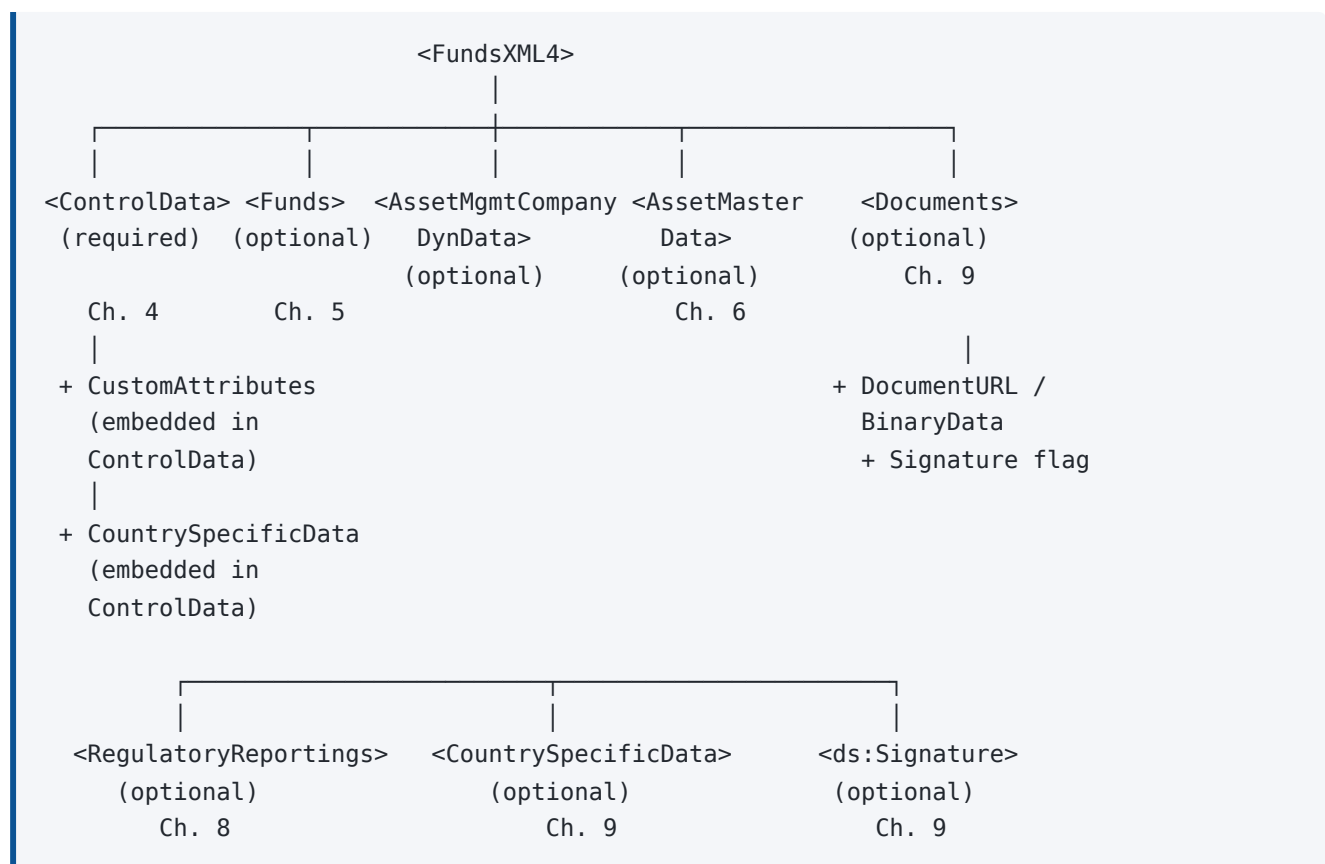
Reading the two country-specific regions, the Austrian section inside `ControlData` is structurally typed: `FundDataPortalContent` is a defined element from `CountrySpecificControlDataATType` in the schema, and `FundDataPortalInfos/Info` has a defined complex type with a required `type` attribute. Every field in the Austrian block is checked against the schema during validation. The Luxembourg section at the root, by contrast, is declared as `xs:anyType` in the local schema, so the producer is free to invent element names like

`CSSFRegistration` and `RegistrationNumber` — the schema accepts them without structural checking, and a consumer that understands the Luxembourg content conventions can parse them out. The document as a whole validates; the difference between the two country branches is in how much *structural* validation each branch imposes on its content.

9.6 How the Advanced Areas Fit Together

The four advanced areas treated in this chapter — `Documents`, `ds:Signature`, `CustomAttributes`, and `CountrySpecificData` — sit alongside the five main areas from Chapters 4 to 8. Figure 9.1 shows the complete set of possible children of the root `FundsXML4` element, in the order the schema requires.

Figure 9.1 — The complete root structure of a FundsXML 4.2.8 document



The figure makes three points. First, **all four advanced areas are optional**: a minimal valid FundsXML document needs only `ControlData`, and every other root child — including the main content in `Funds` and `AssetMasterData` — is optional at the schema level. Second, **`CustomAttributes` appears in two places**: as a last child of `ControlData` (where it typically carries delivery-level operational metadata) and as a last child of several of the main content types (where it carries content-level extensions). The two uses are orthogonal: a producer can populate `CustomAttributes` on any of the element types that allow it without interfering with the others. Third, **the signature, if present, must come last**: the XSD specifies `ds:Signature` as the final optional child of `FundsXML4`, so a document with a signature has the signature after every other section. This is structural, not decorative — it is what allows the enveloped-signature transform to compute correctly.

A production FundsXML delivery that uses all four advanced areas is not common, but it is possible. The Europa Growth Fund's quarterly ESAP submission (Chapter 8) uses three of them: a `Documents` section with the KID and the annual report, a top-level `CountrySpecificData` section with Luxembourg and German extensions for the destination regulators, and an enveloped `ds:Signature` at the root so that the regulator's ingestion pipeline can verify the source. `CustomAttributes` is used more sparingly, only where the Europa Asset Management ingestion pipeline needs to signal an internal metadata value that the schema's first-class fields do not cover.

9.7 Common Pitfalls

The following short list captures the mistakes that, in our experience, cause the greatest share of advanced-area-related production incidents.

- **`Documents` URLs stop resolving after an archive is retrieved.** A consumer archives a FundsXML file but not the PDFs it references, and six months later the URLs return 404. The fix is either to use `BinaryData` for content that must be archived atomically, or to run a consumer-side fetch-and-store step as part of every delivery's ingestion.
- **`Document/Signature=Digital` confused with a root-level `ds:Signature`.** The flag indicates that the *referenced PDF* is signed; the root `ds:Signature` would sign the *FundsXML envelope*. Mistaking the two produces the wrong expectations about what has and has not been cryptographically authenticated.
- **`XMLDSig` validation fails after a "cosmetic" edit.** A producer reformats the signed document (adds white-space, re-orders attributes) without re-signing it, and the downstream verifier rejects the signature. The fix is: never touch a signed document after signing. If a change is needed, re-sign.
- **`CustomAttributes` used for content that has a native schema home.** The fund's management fee, its benchmark name, its ISIN — these all have first-class schema fields, and putting them into `CustomAttributes` breaks consumers that expect to find them at their proper location. Use the native fields.
- **`CustomAttributes` with unnamespaced names.** `status` is a terrible attribute name because it collides with every other producer that uses it. Use `producer.section.name` conventions.
- **A country-specific branch is populated in the wrong place.** A producer puts CSSF registration content inside `ControlData/CountrySpecificData/LU` when it belongs in the root-level `CountrySpecificData/LU`, or vice versa. Read §9.5.1 and pick the right region for the content.
- **`ds:Signature` placed before the content it should sign.** The schema is explicit that `ds:Signature` comes *last* in the sequence of root children. A file that puts the signature earlier does not validate.

9.8 Key Takeaways

- `Documents` is a root-level schema area that carries fund-related documents alongside the structured data. Each `Document` has a typed classification (*Factsheet*, *KID*, *PRIIPS-KID*, *Prospectus*, *AnnualReport*, *AuditReport*, *AIFMD*), language, applicability to fund/share-class, and one of two content modes — `DocumentURL` or `BinaryData`.

- The schema-imported `ds:Signature` element at the root of `FundsXML4` allows enveloped XML digital signatures over the whole document. The signature algorithm, canonicalisation method, and digest method are chosen through standard XMLDSig `Algorithm` URIs, with RSA-SHA-256 and Canonical XML 1.0 as the current defaults.
- `Document/Signature` (the enum flag with values *No*, *Digital*, *Scan*) and the root-level `ds:Signature` are orthogonal: the former signals whether a referenced PDF is signed, the latter signs the FundsXML envelope. A document can have either, both, or neither.
- `CustomAttributes` — not `CustomDataFields` — is the FundsXML extension mechanism. Each `Attribute` has a `Name`, a `Type` (T/N/D/B), and one of `ValueText` / `ValueNumber` / `ValueDate` / `ValueBoolean`. The mechanism appears at many points in the schema and lets producers carry proprietary metadata without forking the standard.
- `CountrySpecificData` appears in two structurally distinct places: inside `ControlData` for country-specific *control-layer* metadata, and at the root of the document for country-specific *content*. Austrian control-layer extensions are strongly typed; most other national regions use `xs:anyType` and accept free-form content.
- Every XML listing in this chapter validates against `FundsXML4.xsd` with `xmllint --noout --schema FundsXML4.xsd <file>`. The listings are self-contained FundsXML documents, not fragments, and can be copied verbatim into a file and validated as-is.

With the advanced schema areas covered, Part II of the book is complete. Chapter 10 opens Part III with a treatment of validation and quality assurance — the two-stage validation model (schema validation plus business validation), the tolerance rules that production pipelines should apply, and a complete validation workflow built around the Europa Growth Fund's monthly delivery. Chapter 11 follows with the principal tools of the FundsXML ecosystem.



PART III — IMPLEMENTATION AND PRACTICE · CHAPTER 10

Validation and Quality Assurance

Ensuring correct FundsXML files

10.1 Setting the Scene: Why Validation Matters

Part II of this book taught the reader to *read and produce* FundsXML documents. Part III, which this chapter opens, is about *running them in production*. The distinction matters: a document that parses cleanly in a text editor is not necessarily a document that will survive a production pipeline. Real deliveries cross network boundaries, pass through multiple systems, and are consumed by consumers whose tolerance for sloppy data ranges from "reject on first error" to "silently ignore and continue". The producer who does not validate every file before emitting it is pushing that tolerance onto someone else.

This chapter treats validation as an engineering discipline rather than an afterthought. It introduces the two-stage validation model that every production FundsXML pipeline should implement, shows how to run the first stage (schema validation) with the free `xmllint` tool and with the four programming languages most commonly found in production pipelines (Python, Java, C# and PowerShell), catalogues the five or six error patterns that account for the overwhelming majority of real-world validation failures, explains why schema validation alone is not enough, introduces Schematron as the standard tool for the second stage (business-rule validation) with runners in each of the same four languages, and assembles a complete validation workflow — in both Bash and PowerShell — that a producer can drop into its delivery pipeline as a gatekeeper. Every command shown in the chapter has been run against real FundsXML files, and every error message has been captured verbatim from the actual tool output.

By the end of this chapter, you should be able to:

- explain the two-stage validation model and name what each stage catches that the other misses;
- run `xmllint --noout --schema FundsXML4.xsd` against any FundsXML file and interpret the results;
- validate a FundsXML file using Python, Java, C# or PowerShell as an alternative to `xmllint`, choosing whichever language matches your production pipeline;
- read an `xmllint` error message and map it to one of the recurring error patterns the chapter catalogues;
- write a Schematron rule that expresses a business constraint the schema cannot enforce;
- compile and execute Schematron rules with the tooling the chapter introduces — in Python, Java, C# or PowerShell;
- assemble a validation gatekeeper script (Bash or PowerShell) that a production pipeline can use to accept or reject every outgoing delivery.

The tools used throughout the chapter are free and ubiquitous. `xmllint` comes with `libxml2` and is installed by default on virtually every Linux distribution, in the macOS developer tools, and in the Windows Subsystem for Linux. The language-specific validators use standard libraries and widely available open-source packages: Python's `lxml`, Java's `javax.xml.validation`, .NET's `System.Xml.Schema`, and PowerShell's access to the same .NET classes. Schematron validation is performed with `lxml.isoschematron` (Python), `ph-schematron` (Java), and the ISO Schematron XSLT stylesheets (C# and PowerShell). Appendix E lists the relevant installation paths for readers on other platforms; Chapter 11 treats the dedicated FundsXML tooling ecosystem (FreeXmlToolkit, Online Schema Viewer, and so on) that some readers may prefer for day-to-day work.

10.2 The Two-Stage Validation Model

The central architectural claim of this chapter is that every production FundsXML pipeline should validate every outgoing document in **two** distinct stages, using two different technologies, checking two different classes of correctness. Combining them into a single stage is possible but costs the pipeline the ability to give clear diagnostics when things go wrong; skipping either stage entirely is a recurring source of production incidents.

Stage 1 is schema validation. The input is the FundsXML file and the XSD schema (`FundsXML4.xsd` plus its included modules). The check is: does the file conform to the *structural* rules that XSD can express? Element names, element order, cardinality (which elements are required, which are optional, which are repeatable), data types (dates must look like dates, numbers must look like numbers, enumerations must take one of their allowed values), attribute types, and the basic hierarchical shape of the document. Stage 1 catches everything that can be expressed in XSD; the pass criterion is binary — the file either conforms or it does not.

Stage 2 is business-rule validation. The input is the same FundsXML file, plus a separate rule set written in Schematron (or in an equivalent rule language). The check is: does the file conform to the *semantic* rules that XSD cannot express? "A `DELETE` operation must carry a `RelatedDocumentIDs` pointing at the delivery being retracted." "A share class's `NumberOfShares` must be a positive integer." "The sum of the position market values in a portfolio must equal the fund's `TotalNetAssetValue` within a rounding tolerance." "A `Document` of type `PRIIPS-KID` must reference at least one share class, not a fund-level identifier only." None of these rules can be expressed in XSD, because they depend on comparisons between elements, on arithmetic, or on knowledge of the business domain that the schema language cannot capture.

The two stages are complementary, and each one catches errors that the other cannot see:

- A file with `DataOperation=UPDATE` (an obsolete enum value — see Chapter 4's retroactive correction) fails **Stage 1**: the enumeration rule catches it immediately. Stage 2 would not catch it because Schematron assumes the underlying structure is schema-valid when it runs.
- A file with `DataOperation=DELETE` and no `RelatedDocumentIDs` passes **Stage 1** (the schema allows `RelatedDocumentIDs` as optional even on `DELETE`) but fails **Stage 2** if the producer has written a Schematron rule that enforces the semantic constraint.

The operational implication is that a production pipeline should run Stage 1 *first* — because its failures are easier to interpret and its runtime is faster — and then run Stage 2 only if Stage 1 passed. Running Stage 2 against a structurally broken file produces cascading Schematron errors that are harder to read than the underlying XSD failures.

The complete workflow in §10.9 codifies this ordering.

Figure 10.1 — The two-stage validation pipeline

```

FundsXML file
  |
  v
+-----+
| Stage 1:      |    -- FAIL --> Report XSD errors, stop
| xmllint --schema |    (structural problem,
| (XSD)         |    see 10.4 for reading)
+-----+
  | PASS
  v
+-----+
| Stage 2:      |    -- FAIL --> Report business errors, stop
| Schematron    |    (semantic problem,
| (business rules) |    see 10.8 for rule catalogue)
+-----+
  | PASS
  v
Delivery accepted; safe to emit

```

10.3 Schema Validation with xmllint

`xmllint` is the command-line XML tool that ships with `libxml2`, the XML library underlying a significant fraction of the open-source XML ecosystem. It can parse, format, query with XPath, and — critically for this chapter — validate XML documents against an XSD schema. For FundsXML it is the default Stage 1 tool because it is free, widely installed, well-documented, and quick; the single command required for validation is short enough to embed in shell scripts, Makefiles, and CI pipelines without overhead.

The command that validates a FundsXML document against the schema is:

```
xmllint --noout --schema FundsXML4.xsd delivery.xml
```

The flags decompose as follows. `--schema FundsXML4.xsd` tells `xmllint` to validate against the named XSD file; it accepts a path, either relative to the current working directory or absolute. `--noout` suppresses the normal output of `xmllint` (which would otherwise dump the parsed document to standard output after validation); since we are interested only in the pass/fail result and any error messages, `--noout` keeps the terminal clean. The positional argument is the FundsXML file to validate.

On success, `xmllint` prints a single line to standard output:

```
delivery.xml validates
```

and exits with code `0`. On failure, it prints one or more error messages to standard error — each one naming the offending line and the nature of the violation — followed by a final line:

```
delivery.xml fails to validate
```

and exits with code `3` (the `libxml2` convention for schema-validation failures; other non-zero exit codes indicate different categories of error, such as missing files or malformed XML). A CI pipeline or shell script can branch on the exit code to decide whether to proceed.

A few practical notes before the error catalogue in §10.4.

Schema location. The command above assumes that `FundsXML4.xsd` is in the current working directory. In production, the schema is almost always stored elsewhere — a shared schema directory, a container volume, a URL — and the command needs to point at it explicitly with a path. An equivalent invocation using an absolute path:

```
xmllint --noout --schema /opt/fundxml/4.2.8/FundsXML4.xsd delivery.xml
```

Included schemas. `FundsXML4.xsd` imports `xmldsig-core-schema.xsd` for the digital-signature element (§9.3). The imported file must be reachable from the main schema file — usually by being in the same directory, because the `import` in the XSD uses a relative path. A pipeline that copies `FundsXML4.xsd` without also copying `xmldsig-core-schema.xsd` will produce a confusing "cannot locate schema" error on any file that uses `ds:Signature`, even though the file itself appears complete.

Validation speed. `xmllint` parses and validates at roughly 50 to 200 megabytes per second on a modern workstation, depending on the document's structural complexity. A typical month-end FundsXML delivery — 5 to 20 megabytes for a mid-sized asset manager — validates in well under one second. The performance is not usually a constraint; the failure diagnostics are the interesting part.

Alternative tools. Other validators exist: Apache Xerces (via the Java command-line `xmlvalidator`), Oracle's XDK, .NET's `XmlReader` with `XmlReaderSettings.Schemas`, Python's `lxml`, Go's `encoding/xml`. All of them validate against the same XSD and should produce equivalent pass/fail results, though their error messages vary. §10.5 presents complete validation scripts in each of the four most common production languages so that the reader can choose the tool that fits their pipeline.

10.4 Reading xmllint Error Messages — A Catalogue

Five classes of error account for the overwhelming majority of real-world FundsXML validation failures. This section walks through each class with a captured `xmllint` error message, an explanation of what the message means, and the fix. Every error shown in this section was produced by a real `xmllint` run against a real (deliberately broken) FundsXML file, and the output is reproduced verbatim.

10.4.1 Invalid Enumeration Value

A producer emits a file with `DataOperation=UPDATE`, a value that was legal in some older conventions (and appeared in early drafts of this book) but that is not in the current FundsXML 4.2.8 enumeration. The schema defines `DataOperation` as an enumerated field with exactly three values: `INITIAL`, `AMEND`, `DELETE`. Running `xmllint` against the file produces:

```
bad-enum.xml:15: element DataOperation: Schemas validity error :  
  Element 'DataOperation': [facet 'enumeration'] The value 'UPDATE' is  
  not an element of the set {'INITIAL', 'AMEND', 'DELETE'}.  
bad-enum.xml fails to validate
```

The message is unusually clear: it names the offending element, the exact line in the file, the nature of the constraint (facet 'enumeration'), the rejected value, and the full list of allowed values. A reader new to `xmllint` might expect every error message to be this helpful; most of them are.

The fix is to replace the enum value. In this case, `UPDATE` should become `AMEND`, following the correction that Chapter 4 was retroactively updated for. A producer pipeline that has shipped files with `UPDATE` in the past also needs to audit any downstream consumer that has been treating the invalid value permissively — production systems that silently coerce unknown enum values to a default (a bad practice, but a common one) may be silently misclassifying deliveries.

10.4.2 Wrong Element Order

The XSD sequence compositor requires children to appear in a specific order. `ControlData`, for example, must have `UniqueDocumentID`, then `DocumentGenerated`, then optionally `Version`, then `ContentDate`, then `DataSupplier`, then further elements — in exactly that sequence. A producer who writes `ContentDate` before `DocumentGenerated` violates the order, even though both elements are present and the data is correct. Running `xmllint`:

```
bad-order.xml:6: element ContentDate: Schemas validity error :  
  Element 'ContentDate': This element is not expected.  
  Expected is ( DocumentGenerated ).  
bad-order.xml fails to validate
```

The message "This element is not expected. Expected is (DocumentGenerated)" is the canonical `xmllint` wording for an order violation. The word *expected* is deliberately precise: the validator was halfway through parsing the `ControlData` sequence, had just finished `UniqueDocumentID`, and was looking for the next element in the sequence — which should have been `DocumentGenerated`. Instead it found `ContentDate`, an element that legally belongs later in the sequence but not yet.

The fix is to reorder the elements to match the XSD's declared sequence. The schema itself is the authoritative source; Appendix C's XSD quick reference lists the sequence for every major type.

10.4.3 Missing Required Child

The XSD distinguishes required children (those with no `minOccurs` attribute, or `minOccurs="1"`) from optional ones (`minOccurs="0"`). A producer who omits a required child — because they thought it was optional, or because their generator had a bug — produces a file that `xmllint` rejects with a "Missing child element(s)" message. A `DataSupplier` block, for example, requires a `Type` element. Omit it and you get:

```
bad-missing.xml:9: element DataSupplier: Schemas validity error :  
  Element 'DataSupplier': Missing child element(s). Expected is ( Type ).  
bad-missing.xml fails to validate
```

The "Expected is (Type)" tells the reader exactly which required child was missing. When several children are missing, `xmllint` reports the *first* missing one it encounters rather than enumerating all of them; fixing that one and re-running the validation often reveals further missing children in subsequent runs. The iteration is mildly annoying but not difficult.

10.4.4 Invalid Type

XSD types enforce lexical constraints on the values of elements. A field declared as `xs:date` must carry a value that looks like `2026-03-31` (ISO 8601 date format); `xs:dateTime` requires a value like `2026-04-01T06:47:13Z`; `xs:integer` requires a string of digits; `xs:boolean` requires one of `true`, `false`, `1`, or `0`. A producer who writes a date in a local format — say, `31/03/2026` — breaks the type constraint:

```
bad-type.xml:8: element ContentDate: Schemas validity error :  
  Element 'ContentDate': '31/03/2026' is not a valid value of the  
  atomic type 'xs:date'.  
bad-type.xml fails to validate
```

The fix is to convert the value to the expected ISO format. This error is particularly common in producers that have been built around European-locale date-picker libraries (which default to `DD/MM/YYYY`) without explicit format conversion at the XML-serialisation step. The producer's output layer should always emit ISO dates regardless of the system's display locale.

10.4.5 Unknown Element

The complement of the "wrong order" and "missing child" errors is the "unknown element" error: a producer emits an element that is not declared in the schema at all, at a position where the schema does not expect any element (or, more commonly, any element with that name). This is the error that revealed the `DataSupplier/LEI` inaccuracy in earlier drafts of this book. A producer who writes an `LEI` child inside `DataSupplier` — following the natural assumption that every organisation has an LEI — sees:

```
bad-unknown.xml:14: element LEI: Schemas validity error :  
  Element 'LEI': This element is not expected. Expected is ( Contact ).  
bad-unknown.xml fails to validate
```

The surprise here is that `DataSupplierType` in the FundsXML 4.2.8 schema has no `LEI` field at all. The `SystemCountry`, `Short`, `Name`, `Type`, and optional `Contact` children are the only ones it accepts. A producer who wants to convey the supplier's LEI has to put it somewhere else — typically through `CustomAttributes` (Chapter 9.4), or through the supplier's `Short` code if the code is in a format that includes the LEI by convention. The lesson is general: *read the schema before assuming what fields exist*, because some fields that feel natural are not actually in the standard.

10.4.6 What xmllint Does Not Catch

`xmllint` catches everything that XSD can express, which is a lot but not everything. Specifically, it does not check:

- Whether a `UniqueDocumentID` is actually unique across a producer's output stream (a cross-file semantic property).
- Whether the sum of portfolio market values matches the fund's total net assets within a tolerance (an arithmetic cross-element property).
- Whether a `DELETE` operation carries the `RelatedDocumentIDs` that the FundsXML documentation recommends (a conditional semantic rule; the schema leaves it optional).
- Whether a `Document` of type `PRIIPS-KID` actually points at an existing share class (a cross-reference property).
- Whether the PAI values in an EET block are consistent with the portfolio composition (a cross-module property).

None of these are XSD-expressible, and none of them are `xmllint`'s fault to miss. They are the subject of Stage 2, which the rest of this chapter develops.

10.5 Schema Validation in Python, Java, C# and PowerShell

While `xmllint` is the reference tool for XSD validation and the one this book recommends for interactive use, production pipelines are often built in general-purpose programming languages. A Java pipeline wants a Java validator; a .NET pipeline wants a C# validator; a Windows operations team wants a PowerShell script. This section presents a complete XSD validation script in each of the four most common production languages. Every script follows the same pattern as the `xmllint` command: it takes a schema path and a file path, reports errors with line numbers, and exits with a non-zero code on failure. Each script can be invoked directly from the command line — as a drop-in replacement for `xmllint` in the validation step — or imported as a module into a larger application.

Table 10.1 — Schema validation commands at a glance

Tool	Command
xmllint	<code>xmllint --noout --schema FundsXML4.xsd delivery.xml</code>
Python	<code>python3 validate-xsd.py FundsXML4.xsd delivery.xml</code>
Java	<code>java ValidateXsd FundsXML4.xsd delivery.xml</code>
C#	<code>dotnet run -- FundsXML4.xsd delivery.xml</code>
PowerShell	<code>.\Validate-XsdSchema.ps1 -SchemaPath FundsXML4.xsd -XmlPath delivery.xml</code>

10.5.1 Python (lxml)

Python's `lxml` library wraps `libxml2` — the same library behind `xmllint` — in a Pythonic API. Schema validation is a two-step process: parse the XSD into an `XMLSchema` object, then call `validate()` on the target document. Errors are available in the schema's `error_log`, with line numbers and messages that closely mirror `xmllint`'s output.

```
#!/usr/bin/env python3
"""validate-xsd.py – XSD schema validation for FundsXML files."""
import sys
from lxml import etree

if len(sys.argv) != 3:
    print("usage: validate-xsd.py <schema.xsd> <file.xml>", file=sys.stderr)
    sys.exit(2)

schema_path, xml_path = sys.argv[1], sys.argv[2]

schema_doc = etree.parse(schema_path)
schema = etree.XMLSchema(schema_doc)
doc = etree.parse(xml_path)

if schema.validate(doc):
    print(f"{xml_path} validates")
    sys.exit(0)

for error in schema.error_log:
    print(f"  line {error.line}: {error.message}")
print(f"{xml_path} fails to validate")
sys.exit(1)
```

The script is self-contained and can be run from the command line with `python3 validate-xsd.py FundsXML4.xsd delivery.xml`. The prerequisite is `lxml`, which is installed via `pip install lxml` (or `apt install python3-lxml` on Debian/Ubuntu).

To use the same logic in a larger application — say, a Django-based upload portal that validates incoming FundsXML deliveries — extract the core into a function:

```
def validate_xsd(schema_path: str, xml_path: str) -> list[str]:
    """Return a list of error messages, empty on success."""
    schema = etree.XMLSchema(etree.parse(schema_path))
    doc = etree.parse(xml_path)
    schema.validate(doc)
    return [str(e) for e in schema.error_log]
```

10.5.2 Java (javax.xml.validation)

Java's standard library includes a full XSD validation API in the `javax.xml.validation` package, available in every JDK since Java 5. No third-party dependency is required. The validator is built from a `SchemaFactory`, which parses the XSD into a `Schema` object, and a `Validator`, which checks the target document against the schema. Errors are reported through an `ErrorHandler` callback.

```
// ValidateXsd.java – XSD schema validation for FundsXML files
import javax.xml.XMLConstants;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class ValidateXsd {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.err.println(
                "usage: java ValidateXsd <schema.xsd> <file.xml>");
            System.exit(2);
        }
        try {
            SchemaFactory factory = SchemaFactory.newInstance(
                XMLConstants.W3C_XML_SCHEMA_NS_URI);
            Schema schema = factory.newSchema(new File(args[0]));
            Validator validator = schema.newValidator();

            List<String> errors = new ArrayList<>();
            validator.setErrorHandler(new ErrorHandler() {
                @Override
                public void warning(SAXParseException e) { }
                @Override
```

```

        public void error(SAXParseException e) {
            errors.add("line " + e.getLineNumber()
                + ": " + e.getMessage());
        }
        @Override
        public void fatalError(SAXParseException e) {
            errors.add("line " + e.getLineNumber()
                + ": " + e.getMessage());
        }
    });

    validator.validate(new StreamSource(new File(args[1])));

    if (errors.isEmpty()) {
        System.out.println(args[1] + " validates");
    } else {
        for (String err : errors) {
            System.out.println("  " + err);
        }
        System.out.println(args[1] + " fails to validate");
        System.exit(1);
    }
} catch (Exception e) {
    System.err.println("error: " + e.getMessage());
    System.exit(2);
}
}
}

```

Compile and run with:

```

javac ValidateXsd.java
java ValidateXsd FundsXML4.xsd delivery.xml

```

No external library is needed — the JDK's built-in Xerces implementation handles the XSD parsing and validation. For larger applications, extract the validation logic into a utility method that returns the list of errors, and call it from wherever the pipeline needs a Stage 1 check.

10.5.3 C# (.NET)

.NET provides XSD validation through the `System.Xml.Schema` namespace. The pattern mirrors the Java approach: load the schema into an `XmlSchemaSet`, configure an `XmlReaderSettings` with that schema set and a `ValidationType` of `Schema`, then read the document through the validating reader. Errors are reported through the `ValidationEventHandler` delegate.

```

// ValidateXsd.cs – XSD schema validation for FundsXML files
using System;
using System.Collections.Generic;

```

```
using System.Xml;
using System.Xml.Schema;

class ValidateXsd
{
    static int Main(string[] args)
    {
        if (args.Length != 2)
        {
            Console.Error.WriteLine(
                "usage: ValidateXsd <schema.xsd> <file.xml>");
            return 2;
        }

        var schemas = new XmlSchemaSet();
        schemas.Add(null, args[0]);

        var settings = new XmlReaderSettings
        {
            Schemas = schemas,
            ValidationType = ValidationType.Schema
        };

        var errors = new List<string>();
        settings.ValidationEventHandler += (sender, e) =>
            errors.Add($"line {e.Exception.LineNumber}: {e.Message}");

        using (var reader = XmlReader.Create(args[1], settings))
        {
            while (reader.Read()) { }
        }

        if (errors.Count == 0)
        {
            Console.WriteLine($"{args[1]} validates");
            return 0;
        }
        foreach (var err in errors)
            Console.WriteLine($" {err}");
        Console.WriteLine($"{args[1]} fails to validate");
        return 1;
    }
}
```

To compile and run as a standalone console application:

```
dotnet new console -n ValidateXsd
# replace Program.cs with the code above
dotnet run --project ValidateXsd -- FundsXML4.xsd delivery.xml
```

No NuGet package is needed — `System.Xml` ships with every .NET installation. For integration into a larger application (an ASP.NET Core service, a WPF desktop tool), the same `XmlSchemaSet / XmlReaderSettings` pattern works as a library call.

10.5.4 PowerShell

PowerShell has native access to the .NET `System.Xml` classes, so XSD validation requires no external module. The script below uses the same `XmlSchemaSet` and `XmlReaderSettings` pattern as the C# example, wrapped in PowerShell syntax. It is the natural choice for Windows operations teams and for pipelines orchestrated with PowerShell scripts.

```
# Validate-XsdSchema.ps1 – XSD schema validation for FundsXML files
# Usage: .\Validate-XsdSchema.ps1 -SchemaPath FundsXML4.xsd -XmlPath delivery.xml
param(
    [Parameter(Mandatory)][string]$SchemaPath,
    [Parameter(Mandatory)][string]$XmlPath
)

$schemaSet = New-Object System.Xml.Schema.XmlSchemaSet
$schemaSet.Add($null, (Resolve-Path $SchemaPath).Path) | Out-Null

$settings = New-Object System.Xml.XmlReaderSettings
$settings.Schemas = $schemaSet
$settings.ValidationType = [System.Xml.ValidationType]::Schema

$script:errors = @()
$handler = [System.Xml.Schema.ValidationEventHandler]{
    param($sender, $e)
    $script:errors += "line $($e.Exception.LineNumber): $($e.Message)"
}
$settings.add_ValidationEventHandler($handler)

$reader = [System.Xml.XmlReader]::Create(
    (Resolve-Path $XmlPath).Path, $settings)
try {
    while ($reader.Read()) { }
}
finally {
    $reader.Close()
}

if ($errors.Count -eq 0) {
    Write-Output "$XmlPath validates"
    exit 0
}
foreach ($err in $errors) {
    Write-Output " $err"
}
```

```
Write-Output "$XmlPath fails to validate"
exit 1
```

Run it from a PowerShell prompt (or from a Windows Terminal session) with:

```
.\Validate-XsdSchema.ps1 -SchemaPath FundsXML4.xsd -XmlPath delivery.xml
```

The exit code convention follows `xmllint`: `0` for success, `1` for validation failure, `2` for usage errors. A CI/CD pipeline running on Windows (Azure DevOps, GitHub Actions with a `windows-latest` runner) can use the exit code to gate subsequent steps.

A note on choosing. All five tools — `xmllint`, Python, Java, C#, PowerShell — validate against the same XSD and will produce the same pass/fail verdict on any given file. The error messages differ in wording but describe the same underlying violations. Choose the tool that matches the language of your pipeline: a Python ETL job should use `lxml`, a Java microservice should use `javax.xml.validation`, a .NET data hub should use `System.Xml.Schema`, and a Windows operations script should use PowerShell. There is no correctness advantage to any of them over the others.

10.6 Business Validation: Where Schema Ends

The limits of Stage 1 are the boundaries of what XSD can express. Those boundaries are well-defined — XSD is a formal specification, and the set of constraints it can enforce is known precisely — but they leave out whole categories of correctness that matter in production. Business validation is the discipline of closing the gap.

Three broad classes of rule live outside XSD's reach.

Class 1: Conditional presence rules. "If X is set, Y must also be set." The schema can express required-versus-optional as a fixed property of an element, but it cannot say "this element is required when some *other* element takes a specific value". The `DataOperation` example from §10.4.6 is the classic case: the enum has three values, and for one of them (`DELETE`) the documentation recommends that `RelatedDocumentIDs` be present. The schema allows `RelatedDocumentIDs` as optional for all three values. Enforcing the conditional rule requires a separate language.

Class 2: Cross-element arithmetic. "The sum of children must equal a parent figure, or must match to within a tolerance." The fund's portfolio has individual position market values, and the fund also has a `TotalNetAssetValue`. In a well-built delivery, the sum of the positions (after currency conversion) should equal the total net assets; a mismatch suggests a data quality problem. XSD cannot express arithmetic over element values; it treats each element as an independent value with its own lexical constraints.

Class 3: Cross-reference integrity. "This identifier must point at a record that exists elsewhere in the file." A portfolio position references an `AssetMasterData` entry through a `UniqueID`; the entry must exist for the reference to be meaningful. A `Document` references a share class through an `ISIN`; the share class must be one that the fund

actually has. XSD has a limited facility for this through `xs:key` and `xs:keyref`, and the FundsXML schema uses it for a few specific cases (we saw one in Chapter 5 — the benchmark ID link between static and dynamic data), but the general case is beyond what XSD can express.

The standard tool for expressing all three classes of rule — together with many others — is **Schematron**, a rule language designed specifically for business validation of XML documents. §10.7 introduces the language; §10.8 shows a library of concrete rules for FundsXML.

10.7 Schematron as a Business-Rule Language

Schematron is an ISO/IEC standard (ISO/IEC 19757-3:2016) for rule-based XML validation. Its design is deliberately simple: a Schematron file is an XML document that contains a set of **rules**, each of which names a **context** (an XPath expression identifying the elements the rule applies to) and a set of **assertions** (further XPath expressions that must evaluate to true when the rule fires). If any assertion fails, Schematron reports the failure together with a human-readable message that the rule author supplied.

The approach is complementary to XSD's structural approach. XSD describes what a valid document *looks like*; Schematron describes what a valid document *satisfies*. Both approaches are necessary, because they catch different kinds of error, and neither can substitute for the other. A Schematron rule running against a structurally broken document produces confusing errors (because the document does not have the shape the XPath expressions assume); an XSD-only pipeline misses all of the semantic rules that XSD cannot express.

10.7.1 The Structure of a Schematron File

A minimal Schematron file has three levels of nesting:

- A top-level `<sch:schema>` element, containing
- one or more `<sch:pattern>` elements (logical groupings of related rules), each containing
- one or more `<sch:rule>` elements, each of which has a `context` attribute and one or more `<sch:assert>` children.

A single rule looks like this:

```
<sch:rule context="ControlData">
  <sch:assert test="DataOperation != 'DELETE' or RelatedDocumentIDs">
    A DELETE operation must name the delivery being retracted in
    RelatedDocumentIDs. Delivery <sch:value-of select="UniqueDocumentID"/>
    violates this rule.
  </sch:assert>
</sch:rule>
```

The `context="ControlData"` attribute tells Schematron: "for every `ControlData` element anywhere in the document, run the assertions inside this rule." The `test` attribute of `<sch:assert>` is an XPath expression that must evaluate to `true` for the assertion to pass. If the expression evaluates to `false`, Schematron reports the con-

tained text as a failure message, substituting any `<sch:value-of>` elements with the corresponding XPath values from the document.

The rule above reads informally: *for each `ControlData`, either the `DataOperation` is not `DELETE`, or a `RelatedDocumentIDs` sibling element exists*. This is the logically equivalent rewriting of "if `DataOperation` is `DELETE`, then `RelatedDocumentIDs` must be present"; the disjunction form is how conditional rules are typically expressed in Schematron, because the XPath language does not have a direct "if-then" operator at the predicate level.

10.7.2 Running Schematron with Python

Schematron rules are not executed directly. The canonical execution model is to **compile** the Schematron file into an XSLT stylesheet, then apply the stylesheet to the target XML document. The output is an SVRL (Schematron Validation Report Language) document listing each failed assertion with its location and message. This two-step pipeline — Schematron compilation followed by XSLT execution — is standardised, portable, and supported by every major XML toolchain.

For Python, we use `lxml.isoschematron`, which wraps the compilation and execution in a single API call. The module is free, widely installed, and produces results equivalent to any XSLT-based Schematron processor. A minimal runner script:

```
#!/usr/bin/env python3
"""run-schematron.py — Run ISO Schematron rules against a FundsXML file."""
import sys
from lxml import etree
from lxml.isoschematron import Schematron

if len(sys.argv) != 3:
    print("usage: run-schematron.py <rules.sch> <file.xml>", file=sys.stderr)
    sys.exit(2)

rules_path, xml_path = sys.argv[1], sys.argv[2]

rules_doc = etree.parse(rules_path)
schematron = Schematron(rules_doc, store_report=True)

xml_doc = etree.parse(xml_path)
ok = schematron.validate(xml_doc)

report = schematron.validation_report
failed_asserts = report.findall(
    ".//{http://purl.oclc.org/dsdl/svrl}failed-assert"
)

if ok and not failed_asserts:
    print(f"{xml_path}: Schematron validation passed")
    sys.exit(0)

print(f"{xml_path}: Schematron validation FAILED")
```

```

for fa in failed_asserts:
    loc = fa.get("location", "?")
    text_node = fa.find("{http://purl.oclc.org/dsdl/svrl}text")
    msg = " ".join(text_node.text.split()) if text_node is not None else "(no message)"
    print(f"    at {loc}")
    print(f"        {msg}")
sys.exit(1)

```

The script reads a Schematron rule file and an XML document from the command line, compiles the rules, runs them against the document, and prints either a success line or a list of failed assertions with their locations and messages. Exit code `0` signals success, `1` signals business-rule failures, `2` signals argument errors. Production pipelines can use this exit code exactly as they use `xmllint`'s exit code to decide whether to proceed.

A note on the query binding. Schematron supports several XPath versions through its `queryBinding` attribute on the `<sch:schema>` element; the two we care about are `xslt` (XSLT 1.0, which means XPath 1.0) and `xslt2` (XSLT 2.0, which means XPath 2.0 and a richer expression language). `lxml.isoschematron` supports only the default `xslt` binding, so the rules in this chapter are written in XPath 1.0. This is occasionally awkward — XPath 1.0 has no sequence literals, for example, so a membership test must be written as a `contains()` expression over a space-separated string — but is otherwise adequate for the rules most producers need. Readers who prefer XPath 2.0 can substitute an XSLT-2.0-capable processor such as Saxon, at the cost of a heavier dependency.

10.7.3 Running Schematron with Java

Java's most widely used Schematron library is **ph-schematron** by Philip Helger, the same library that powers Schematron validation in the European e-invoicing (EN 16931) ecosystem. It provides a pure-Java implementation that compiles and executes Schematron rules without an external XSLT processor, producing SVRL output through a clean API. The library is available as a Maven dependency:

```

<dependency>
  <groupId>com.helger.schematron</groupId>
  <artifactId>ph-schematron-pure</artifactId>
  <version>8.0.2</version><!-- check for latest release -->
</dependency>

```

A minimal runner:

```

// ValidateSchematron.java – Schematron validation using ph-schematron
import com.helger.schematron.pure.SchematronResourcePure;
import com.helger.schematron.svrl.SVRLHelper;
import com.helger.schematron.svrl.SVRLFailedAssert;
import com.helger.schematron.svrl.jaxb.SchematronOutputType;
import javax.xml.transform.stream.StreamSource;
import java.io.File;
import java.util.List;

public class ValidateSchematron {
    public static void main(String[] args) throws Exception {

```

```

        if (args.length != 2) {
            System.err.println(
                "usage: java ValidateSchematron <rules.sch> <file.xml>");
            System.exit(2);
        }

        SchematronResourcePure sch =
            SchematronResourcePure.fromFile(args[0]);
        if (!sch.isValidSchematron()) {
            System.err.println("Invalid Schematron file: " + args[0]);
            System.exit(2);
        }

        SchematronOutputType svrl =
            sch.applySchematronValidationToSVRL(
                new StreamSource(new File(args[1])));
        List<SVRLFailedAssert> failures =
            SVRLHelper.getAllFailedAssertions(svrl);

        if (failures.isEmpty()) {
            System.out.println(
                args[1] + ": Schematron validation passed");
        } else {
            System.out.println(
                args[1] + ": Schematron validation FAILED");
            for (SVRLFailedAssert fa : failures) {
                System.out.println("    at " + fa.getLocation());
                System.out.println("        " + fa.getText());
            }
            System.exit(1);
        }
    }
}

```

Build and run via Maven (or Gradle):

```

mvn compile exec:java \
    -Dexec.mainClass="ValidateSchematron" \
    -Dexec.args="egf-rules.sch delivery.xml"

```

The `SchematronResourcePure` class handles both the compilation (Schematron to an internal rule model) and the execution (evaluation against the target document) in a single `applySchematronValidationToSVRL` call. The `SVRLHelper.getAllFailedAssertions()` method extracts failed assertions from the SVRL output, each carrying a location string and a human-readable text message — the same information the Python runner prints.

For larger applications (a Spring Boot microservice, a Kafka consumer), extract the validation into a service method that returns the list of `SVRLFailedAssert` objects, and let the caller decide how to format the results.

10.7.4 Running Schematron with C# and PowerShell

The .NET ecosystem has no single dominant Schematron library comparable to Python's `lxml.isoschematron` or Java's `ph-schematron`. The standard approach is to use the **ISO Schematron XSLT stylesheets** — the reference implementation maintained at `github.com/Schematron/schematron` — to compile a `.sch` file into an XSLT stylesheet, then apply that stylesheet to the target document using .NET's built-in `XslCompiledTransform`. The result is an SVRL document that the code parses for failed assertions.

The process has two steps:

1. **Compile:** transform the `.sch` file using `iso_svrl_for_xslt1.xsl` — produces a validation XSLT.
2. **Validate:** apply the validation XSLT to the target XML — produces an SVRL report.

For simple Schematron files (no abstract patterns, no includes — like the rule file in §10.8), the single `iso_svrl_for_xslt1.xsl` stylesheet suffices. Complex Schematron files that use abstract patterns or `sch:include` require two preliminary steps (`iso_dsdl_include.xsl` and `iso_abstract_expand.xsl`); the ISO repository documents the full pipeline.

C# implementation:

```
// ValidateSchematron.cs – Schematron validation via ISO XSLT stylesheets
using System;
using System.IO;
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;

class ValidateSchematron
{
    static int Main(string[] args)
    {
        if (args.Length < 2 || args.Length > 3)
        {
            Console.Error.WriteLine(
                "usage: ValidateSchematron <rules.sch> <file.xml>"
                + " [iso-xslt-dir]");
            return 2;
        }
        string schPath = args[0], xmlPath = args[1];
        string isoDir = args.Length == 3 ? args[2] : ".";

        // Step 1: compile Schematron rules to XSLT
        string isoXslt = Path.Combine(isoDir,
            "iso_svrl_for_xslt1.xsl");
        var compiler = new XslCompiledTransform();
        compiler.Load(isoXslt);

        var compiledStream = new MemoryStream();
        using (var schReader = XmlReader.Create(schPath))
        using (var writer = XmlWriter.Create(compiledStream))
```

```

        compiler.Transform(schReader, writer);
        compiledStream.Position = 0;

        // Step 2: apply compiled XSLT to the target document
        var validator = new XslCompiledTransform();
        using (var xslReader = XmlReader.Create(compiledStream))
            validator.Load(xslReader);

        var svrlStream = new MemoryStream();
        using (var xmlReader = XmlReader.Create(xmlPath))
        using (var writer = XmlWriter.Create(svrlStream))
            validator.Transform(xmlReader, writer);
        svrlStream.Position = 0;

        // Step 3: parse SVRL for failed assertions
        var nav = new XPathDocument(svrlStream)
            .CreateNavigator();
        var mgr = new XmlNamespaceManager(nav.NameTable);
        mgr.AddNamespace("svrl",
            "http://purl.oclc.org/dsdl/svrl");

        var fails = nav.Select(
            "//svrl:failed-assert", mgr);
        if (fails.Count == 0)
        {
            Console.WriteLine(
                $"{xmlPath}: Schematron validation passed");
            return 0;
        }
        Console.WriteLine(
            $"{xmlPath}: Schematron validation FAILED");
        while (fails.MoveNext())
        {
            string loc = fails.Current
                .GetAttribute("location", "");
            var text = fails.Current
                .SelectSingleNode("svrl:text", mgr);
            string msg = text?.Value?.Trim()
                ?? "(no message)";
            Console.WriteLine($"  at {loc}");
            Console.WriteLine($"    {msg}");
        }
        return 1;
    }
}

```

PowerShell implementation:

The same three-step pipeline, using PowerShell's access to the .NET XML and XSLT classes:

```

# Validate-Schematron.ps1 – Schematron validation via ISO XSLT stylesheets
# Usage: .\Validate-Schematron.ps1 -RulesPath egf-rules.sch -XmlPath delivery.xml
param(
    [Parameter(Mandatory)][string]$RulesPath,
    [Parameter(Mandatory)][string]$XmlPath,
    [string]$IsoXsltDir = "."
)

$ErrorActionPreference = "Stop"

# Step 1: compile Schematron rules to XSLT
$isoXslt = Join-Path $IsoXsltDir "iso_svrl_for_xslt1.xsl"
$compiler = New-Object System.Xml.Xsl.XslCompiledTransform
$compiler.Load($isoXslt)

$compiledStream = New-Object System.IO.MemoryStream
$schReader = [System.Xml.XmlReader]::Create(
    (Resolve-Path $RulesPath).Path)
$writer = [System.Xml.XmlWriter]::Create($compiledStream)
try { $compiler.Transform($schReader, $writer) }
finally { $writer.Close(); $schReader.Close() }
$compiledStream.Position = 0

# Step 2: apply compiled XSLT to the target document
$validator = New-Object System.Xml.Xsl.XslCompiledTransform
$xslReader = [System.Xml.XmlReader]::Create($compiledStream)
try { $validator.Load($xslReader) }
finally { $xslReader.Close() }

$svrlStream = New-Object System.IO.MemoryStream
$xmlReader = [System.Xml.XmlReader]::Create(
    (Resolve-Path $XmlPath).Path)
$svrlWriter = [System.Xml.XmlWriter]::Create($svrlStream)
try { $validator.Transform($xmlReader, $svrlWriter) }
finally { $svrlWriter.Close(); $xmlReader.Close() }
$svrlStream.Position = 0

# Step 3: parse SVRL for failed assertions
$doc = New-Object System.Xml.XPath.XPathDocument($svrlStream)
$nav = $doc.CreateNavigator()
$mgr = New-Object System.Xml.XmlNamespaceManager($nav.NameTable)
$mgr.AddNamespace("svrl", "http://purl.oclc.org/dsdl/svrl")

$fails = $nav.Select("//svrl:failed-assert", $mgr)
if ($fails.Count -eq 0) {
    Write-Output "${XmlPath}: Schematron validation passed"
    exit 0
}
Write-Output "${XmlPath}: Schematron validation FAILED"
while ($fails.MoveNext()) {
    $loc = $fails.Current.GetAttribute("location", "")

```

```

$textNode = $fails.Current.SelectSingleNode("svrl:text", $mgr)
$msg = if ($textNode) { $textNode.Value.Trim() } else { "(no message)" }
Write-Output "    at $loc"
Write-Output "        $msg"
}
exit 1

```

Table 10.2 — Schematron validation commands at a glance

Language	Command
Python	<code>python3 run-schematron.py egf-rules.sch delivery.xml</code>
Java	<code>java ValidateSchematron egf-rules.sch delivery.xml</code>
C#	<code>dotnet run -- egf-rules.sch delivery.xml</code>
PowerShell	<code>.\Validate-Schematron.ps1 -RulesPath egf-rules.sch -XmlPath delivery.xml</code>

The C# and PowerShell implementations require one external file — `iso_svrl_for_xslt1.xsl` from the ISO Schematron reference implementation — which is a small (roughly 70 KB), stable, freely available XSLT stylesheet that rarely changes between ISO revisions. Download it once, place it alongside the rule file, and point the script at its directory. In a production deployment, the stylesheet and the compiled Schematron XSLT can be cached so that the compilation step runs only when the rule file changes.

10.8 A Library of Business Rules for FundsXML

This section presents a small but realistic library of Schematron rules that a production FundsXML pipeline might enforce before emitting a delivery. Every rule has been tested against real FundsXML files using the runner from §10.7.2, and the rule file shown below validates cleanly against its own intent: it passes for a correct delivery, and it fails with the expected messages for a broken one. The library is deliberately short — six rules in two patterns — so that the pedagogical point is visible without drowning in detail; production pipelines typically have several dozen rules organised into a larger number of patterns.

The complete rule file:

```

<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:title>Europa Growth Fund – FundsXML business rules</sch:title>

  <sch:pattern id="delivery-semantics">
    <sch:title>ControlData semantic rules</sch:title>

    <sch:rule context="ControlData">

```



```

    <sch:assert test="DataOperation != 'DELETE' or RelatedDocumentIDs">
      A DELETE operation must name the delivery being retracted in
RelatedDocumentIDs.
      Delivery <sch:value-of select="UniqueDocumentID"/> violates this rule.
    </sch:assert>
    <sch:assert test="DataOperation != 'AMEND' or RelatedDocumentIDs">
      An AMEND operation should reference the delivery it amends through
RelatedDocumentIDs.
      Delivery <sch:value-of select="UniqueDocumentID"/> violates this rule.
    </sch:assert>
    <sch:assert test="number(substring(DocumentGenerated, 1, 4)) >=
number(substring(ContentDate, 1, 4))">
      DocumentGenerated (<sch:value-of select="DocumentGenerated"/>)
      cannot be in an earlier year than ContentDate (<sch:value-of
select="ContentDate"/>).
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern id="producer-identity">
  <sch:title>Producer identity rules</sch:title>

  <sch:rule context="DataSupplier">
    <sch:assert test="string-length(Short) >= 2">
      DataSupplier/Short must be at least 2 characters; found "<sch:value-of
select="Short"/>".
    </sch:assert>
    <sch:assert test="contains(' AT DE CH LU FR IT ES NL BE PT SE DK NO FI GB IE ',
concat(' ', SystemCountry, ' '))">
      DataSupplier/SystemCountry "<sch:value-of select="SystemCountry"/>"
      is outside the set of countries supported by this producer.
    </sch:assert>
  </sch:rule>
</sch:pattern>

</sch:schema>

```

10.8.1 Reading the Rules One by One

Rule 1 — DELETE requires RelatedDocumentIDs. The assertion `DataOperation != 'DELETE' or RelatedDocumentIDs` is the classic conditional-presence rewriting. In plain English: for every `ControlData`, it is either the case that the operation is not `DELETE`, or the `RelatedDocumentIDs` element is present. Both conditions fail only when the operation is `DELETE` and `RelatedDocumentIDs` is missing — exactly the case we want to catch. The failure message uses `<sch:value-of select="UniqueDocumentID"/>` to inject the offending document's ID into the error text, which helps a human operator identify the file without having to cross-reference line numbers.

Rule 2 — AMEND should reference the prior delivery. Structurally identical to Rule 1, but applies to `AMEND` rather than `DELETE`. This codifies the recommendation from Chapter 4 that every `AMEND` should point at the delivery it amends through `RelatedDocumentIDs`. The schema leaves this optional; the business rule makes it mandatory for the producer's own deliveries.

Rule 3 — DocumentGenerated year cannot precede ContentDate year. This is an arithmetic rule that XPath 1.0 can express only approximately. The full rule would be "the full `DocumentGenerated` date-time must be on or after the full `ContentDate` date", but XPath 1.0 has no date arithmetic, so we compare the year substrings as numbers. The approximation catches the common case (a pipeline that has been configured with the wrong content-date default and produces deliveries dated years in the past) but misses the edge case where the two dates differ within the same year. A production pipeline that needs the full check can either use an XSLT-2.0 Schematron binding with `xs:dateTime` comparison, or accept the approximation as an 80%-solution.

Rule 4 — DataSupplier/Short must be at least 2 characters. A lexical rule that XSD's `Text64Type` does not enforce. The rule reflects a project convention that single-character identifiers are too short to be useful and almost always indicate a default value that the producer forgot to replace. The failure message includes the offending value.

Rule 5 — DataSupplier/SystemCountry must be in an allowlist. An enumerated-membership rule that XSD's `ISOCountryCodeType` does not enforce: the ISO type accepts every valid country code in the world, but the Europa Growth Fund's pipeline only serves European distribution countries. The rule restricts the allowed set to the producer's actual footprint. The XPath 1.0 implementation uses `contains()` over a space-delimited string because there are no sequence literals in the query language.

10.8.2 Running the Rules

Saving the rule file as `egf-rules.sch` and the runner script as `run-schematron.py`, the full two-stage validation of a file looks like this:

```
$ xmllint --noout --schema FundsXML4.xsd good.xml
good.xml validates

$ python3 run-schematron.py egf-rules.sch good.xml
good.xml: Schematron validation passed
```

Both stages pass, and the file is safe to emit. A deliberately broken file — one that uses `DataOperation=DELETE` without a `RelatedDocumentIDs`, and a `DataSupplier/Short` of only one character — produces a different picture:

```
$ xmllint --noout --schema FundsXML4.xsd bad-business.xml
bad-business.xml validates

$ python3 run-schematron.py egf-rules.sch bad-business.xml
bad-business.xml: Schematron validation FAILED
  at /FundsXML4/ControlData
    A DELETE operation must name the delivery being retracted in
    RelatedDocumentIDs. Delivery EGF-20260331-VAL-006 violates this rule.
```

```
at /FundsXML4/ControlData/DataSupplier
    DataSupplier/Short must be at least 2 characters; found "X".
```

The demonstration is the central point of this chapter: the broken file **passes XSD validation** — because neither of the rules can be expressed in XSD — but **fails business validation** with two specific, actionable error messages, each carrying enough context (the document ID, the offending short code) that a human operator can fix the producer immediately. Without the Schematron stage, the broken file would have been emitted, consumed downstream, and potentially caused a production incident before anyone noticed that the `DELETE` had nothing to delete.

The same test with the Java, C# or PowerShell runners produces equivalent output — the error messages come from the Schematron rules themselves, so any standards-compliant processor will report the same text and the same locations.

10.9 A Complete Validation Workflow

The tools of this chapter — `xmllint` (or any of the language-specific validators), the Schematron runners, and a thin wrapper script — can be assembled into a complete validation gatekeeper that a producer pipeline invokes before emitting every delivery. The workflow has three stages: parse (is the XML well-formed?), validate structure (stage 1), and validate business rules (stage 2). Each stage short-circuits the next on failure, and the script exits non-zero on any failure so that upstream callers can treat a non-zero exit as "do not emit this file".

10.9.1 Bash (Linux / macOS / WSL)

```
#!/usr/bin/env bash
# validate.sh – two-stage FundsXML validation gatekeeper
# Usage: validate.sh <delivery.xml>
set -euo pipefail

if [ $# -ne 1 ]; then
    echo "usage: $0 <delivery.xml>" >&2
    exit 2
fi

XML_FILE="$1"
SCHEMA_FILE="${FUNDSXML_SCHEMA:-/opt/fundxml/4.2.8/FundsXML4.xsd}"
RULES_FILE="${FUNDSXML_RULES:-/opt/fundxml/4.2.8/egf-rules.sch}"
SCHEMATRON_RUNNER="${FUNDSXML_SCHEMATRON_RUNNER:-/opt/fundxml/run-schematron.py}"

# Stage 0: well-formedness (this is cheap and gives the clearest error
# when the file is fundamentally broken).
if ! xmllint --noout "$XML_FILE" 2>&1; then
    echo "FAIL: $XML_FILE is not well-formed XML" >&2
    exit 11
fi
```

```
# Stage 1: schema validation against FundsXML4.xsd.
if ! xmllint --noout --schema "$SCHEMA_FILE" "$XML_FILE" 2>&1; then
    echo "FAIL: $XML_FILE failed XSD validation (stage 1)" >&2
    exit 12
fi

# Stage 2: Schematron business-rule validation.
if ! python3 "$SCHEMATRON_RUNNER" "$RULES_FILE" "$XML_FILE"; then
    echo "FAIL: $XML_FILE failed Schematron validation (stage 2)" >&2
    exit 13
fi

echo "OK: $XML_FILE passed both validation stages"
exit 0
```

10.9.2 PowerShell (Windows)

The following script is the PowerShell equivalent of the Bash gatekeeper above. It uses .NET's `System.Xml` for well-formedness and XSD validation (Stage 0 and Stage 1), and calls the Python Schematron runner for Stage 2. A purely native PowerShell pipeline can replace the Python call with the `Validate-Schematron.ps1` script from §10.7.4.

```
# Validate-Pipeline.ps1 – two-stage FundsXML validation gatekeeper
# Usage: .\Validate-Pipeline.ps1 -XmlFile delivery.xml
param(
    [Parameter(Mandatory)][string]$XmlFile,
    [string]$SchemaFile = "C:\fundxml\4.2.8\FundsXML4.xsd",
    [string]$RulesFile = "C:\fundxml\4.2.8\egf-rules.sch",
    [string]$SchematronRunner = "C:\fundxml\run-schematron.py"
)

$ErrorActionPreference = "Stop"

# Stage 0: well-formedness
try {
    $null = [xml](Get-Content -Raw $XmlFile)
    Write-Output "$XmlFile is well-formed"
}
catch {
    Write-Error "FAIL: $XmlFile is not well-formed XML – $_"
    exit 11
}

# Stage 1: schema validation using .NET XmlReader
$schemas = New-Object System.Xml.Schema.XmlSchemaSet
$schemas.Add($null, (Resolve-Path $SchemaFile).Path) | Out-Null

$settings = New-Object System.Xml.XmlReaderSettings
$settings.Schemas = $schemas
```

```

$settings.ValidationType = [System.Xml.ValidationType]::Schema

$script:xsdErrors = @()
$settings.add_ValidationEventHandler({
    param($sender, $e)
    $script:xsdErrors += $e.Message
})

$reader = [System.Xml.XmlReader]::Create(
    (Resolve-Path $XmlFile).Path, $settings)
try { while ($reader.Read()) { } }
finally { $reader.Close() }

if ($xsdErrors.Count -gt 0) {
    foreach ($err in $xsdErrors) { Write-Output " $err" }
    Write-Error "FAIL: $XmlFile failed XSD validation (stage 1)"
    exit 12
}
Write-Output "$XmlFile passes XSD validation"

# Stage 2: Schematron business-rule validation
& python3 $SchematronRunner $RulesFile $XmlFile
if ($LASTEXITCODE -ne 0) {
    Write-Error "FAIL: $XmlFile failed Schematron validation (stage 2)"
    exit 13
}

Write-Output "OK: $XmlFile passed both validation stages"
exit 0

```

Both scripts use the same exit-code convention: `2` for invocation errors, `11` for well-formedness failures, `12` for XSD failures, `13` for Schematron failures, `0` for success. An upstream pipeline that sees `12` knows that the failure was structural and can route the file to the producer for immediate re-generation; a pipeline that sees `13` knows the failure was semantic and can route the file to a producer-side business-rule review workflow.

The environment variables in the Bash script (`FUNDXML_SCHEMA`, `FUNDXML_RULES`, `FUNDXML_SCHEMATRON_RUNNER`) and the parameter defaults in the PowerShell script let deployers override the paths without editing the script. In a typical deployment, these are set in the pipeline orchestrator's configuration and point at versioned schema and rule files that live alongside the pipeline code.

A production variant of either script would also log its results to a structured audit log (so that incident investigation later can find every validation failure), might run Stage 1 and Stage 2 in parallel for speed on large deliveries (though the serial version shown above is easier to reason about), and could apply differential rules for different delivery categories (ESAP submissions might use a stricter rule set than bilateral distributor files). None of these variations change the underlying two-stage model; they refine how the pipeline is wired around it.

Fully native alternatives. The PowerShell script above calls Python for Stage 2. A team that wants to avoid a Python dependency on its Windows servers can replace the Python call with the native PowerShell Schematron validator from §10.7.4, or with the Java runner from §10.7.3 invoked via `java -jar`. Similarly, the Bash script's

Stage 1 can be replaced with the Python XSD validator from §10.5.1 if `xmllint` is not available. The validation logic is the same regardless of the tool; what changes is the runtime dependency.

10.10 Common Pitfalls

- **Running Stage 2 without Stage 1.** A pipeline that skips XSD validation on the assumption that Schematron will catch everything gets confusing error cascades when the underlying structure is broken. Schematron's XPath expressions assume the document has the shape they describe; a missing `ControlData` produces an empty match set rather than a clear "ControlData missing" error.
 - **Relying on Stage 1 alone.** The complementary mistake: skipping Schematron and hoping that XSD validation is enough. It is not. Every business rule from §10.6 — conditional presence, cross-element arithmetic, cross-reference integrity — lives outside XSD and is caught only by Stage 2. A pipeline that passes Stage 1 but has no Stage 2 still ships files with the kind of semantic bugs that cause production incidents.
 - **Using the wrong schema version.** `xmllint` validates against whichever schema you give it. Validating a FundsXML 4.2.8 file against a 4.2.3 schema may either pass (if the file uses only 4.2.3 features) or fail with confusing "element not expected" errors (if it uses any of the fields that were added between 4.2.3 and 4.2.8). Keep a single canonical schema version in the pipeline, versioned in lockstep with the producer's generator.
 - **Copying FundsXML4.xsd without xmldsig-core-schema.xsd.** Any document with a `ds:Signature` will fail validation with a misleading "cannot locate schema" error that does not immediately point at the missing dependency. Copy both files together.
 - **Treating Schematron failures as warnings rather than errors.** A validation stage whose failures do not block emission is not a validation stage; it is a logging stage. If a rule is worth writing, it is worth enforcing. If a rule is too aggressive to enforce, either relax the rule or remove it from the pipeline entirely — do not run it as a "warning".
 - **Forgetting to validate the validation.** Schematron rules are themselves XML files, and they can contain bugs (incorrect XPath expressions, typos in context selectors, assertions that are always true). A mature pipeline includes unit tests for every rule: a small set of deliberately-broken fixture files that each rule is expected to catch, run automatically every time the rule file changes. Without those tests, the rule file quietly drifts away from its intent.
 - **Mixing validators across stages.** Using `xmllint` on a Linux build server and .NET `XmlReader` on a Windows staging server is fine — but be aware that error *messages* differ between tools, even though the pass/fail verdict is the same. Automated log parsers that match specific `xmllint` error patterns will break if the pipeline switches to a different validator. Parse exit codes, not error messages, in automation.
-

10.11 Key Takeaways

- Every production FundsXML pipeline should validate every outgoing document in two distinct stages: **Stage 1** is schema validation against the XSD (expressing structural rules), **Stage 2** is business-rule validation against a Schematron rule file (expressing semantic rules). The two stages catch disjoint classes of error and are both

necessary.

- **Stage 1** can be executed with `xmllint --noout --schema FundsXML4.xsd delivery.xml`, or with equivalent validators in Python (`lxml`), Java (`javax.xml.validation`), C# (`System.Xml.Schema`), or PowerShell (the same .NET classes). All five tools validate against the same XSD and produce the same pass/fail verdict; choose the one that matches your pipeline's language. §10.4 catalogues the five most common error patterns with captured examples.
- **Stage 2** is executed with Schematron rules compiled from a `.sch` file. Python uses `lxml.isoschematron`; Java uses `ph-schematron`; C# and PowerShell use the ISO Schematron XSLT stylesheets with .NET's `XslCompiledTransform`. Schematron expresses conditional presence rules, cross-element arithmetic, cross-reference integrity, and enumerated-membership constraints that XSD cannot capture. The rules are written as XPath assertions inside `<sch:rule context="...">` blocks and produce human-readable failure messages with document context via `<sch:value-of>`.
- A complete validation gatekeeper script (§10.9) combines well-formedness checking, XSD validation, and Schematron validation into a single wrapper — available in both **Bash** (for Linux, macOS, WSL) and **PowerShell** (for Windows) — with distinct exit codes for each failure category, so that upstream pipelines can route failures to the right remediation workflow.
- The most common pitfalls are running either stage in isolation (each has known blind spots), using the wrong schema version, forgetting to copy the imported XMLDSig schema alongside the main XSD, mixing validators without accounting for error-message differences, and running Schematron rules without unit tests — the last is a subtle failure mode that quietly accumulates drift.

With validation in hand, the next question is: *what tools can we use to produce, inspect, and explore FundsXML files more efficiently than by hand?* Chapter 11 answers that question. It presents the principal tools of the FundsXML ecosystem — FreeXmlToolkit, the Online Schema Viewer, the CSV Converter, and the IDE integrations for IntelliJ, Visual Studio Code, and Eclipse — that complement the free command-line tools this chapter has introduced.



PART III — IMPLEMENTATION AND PRACTICE · CHAPTER 11

Tools and Toolchain

The right tools for productive use

11.1 Setting the Scene: From Command Line to Workstation

Chapter 10 introduced a validation discipline built on two command-line tools: `xmllint` for schema validation and a Python script with `lxml.isoschematron` for business rules. Those tools are free, widely installed, and well-suited to production pipelines where every step needs to be scripted, auditable, and deterministic. They are, however, not well-suited to the other half of FundsXML work: *exploring* a schema interactively, *debugging* a broken delivery by eye, *generating* sample files for testing, *transforming* FundsXML into fact-sheet HTML, or *integrating* FundsXML validation into the editor where a developer or fund-operations analyst already spends their day.

This chapter surveys the tools that fill those interactive and developer-oriented gaps. It treats **FreeXmlToolkit** in detail as the primary desktop workstation for FundsXML work, then covers the smaller and more specialised tools around it (the Online Schema Viewer, the CSV Converter, the FundsXML Generator) and the integration paths into three mainstream IDEs (IntelliJ IDEA, Visual Studio Code, Eclipse). The chapter is deliberately pragmatic: it tells the reader what each tool is for, when to reach for it, and what its typical workflows look like, rather than providing a feature-by-feature reference manual. The tools themselves evolve faster than any book can track; the reference documentation at each tool's website is the authoritative source for the latest features.

One honesty note on the chapter's approach. Unlike Chapter 10, where every command and every error message in the text was produced by running real software on real files, the descriptions in this chapter are grounded in the tools' published documentation and in a reading of their source code where available, but the specific menu paths, button labels, and UI behaviours are as of a point in time and may shift with newer releases. Readers who want to follow along by hand should download the current version of whichever tool they plan to use and cross-check against its own in-application help. Appendix E lists the current official download URLs.

By the end of this chapter, you should be able to:

- pick the right tool for a given FundsXML task — interactive editing, schema browsing, validation, transformation, generation, conversion — and know which tools overlap with which;
 - install FreeXmlToolkit, open a FundsXML document, and use its main modules;
 - navigate the FundsXML schema through the Online Schema Viewer without downloading anything;
 - convert between FundsXML and CSV or Excel for data-entry and data-review workflows;
 - generate sample FundsXML files from the schema for consumer testing;
 - configure IntelliJ IDEA, Visual Studio Code, and Eclipse to validate FundsXML files inline.
-

11.2 A Tour of the FundsXML Ecosystem

Before we look at any single tool, a short map of the landscape is useful, because the tools overlap in ways that the TOC-style listing does not make obvious. FundsXML work falls into a handful of distinct workflows, and most of the tools in the ecosystem serve one or more of them.

Workflow 1 — Reading and exploring a FundsXML file. The user has received a delivery and wants to look at it: browse its structure, verify that particular fields are populated, check whether the schema validation passes, and maybe run a few XPath queries. The primary tool is a FundsXML-aware XML editor, which for most readers means **FreeXmlToolkit** (for desktop use) or one of the **IDE integrations** (for readers who already live in IntelliJ, VS Code, or Eclipse).

Workflow 2 — Exploring the schema itself. The user wants to understand what a particular element means, what its allowed values are, and how it fits into the larger schema hierarchy. For this, the **Online Schema Viewer** is the quickest entry point — it runs in a browser, requires no installation, and presents the FundsXML schema as an explorable, hyperlinked structure. FreeXmlToolkit also offers schema browsing and can generate standalone HTML documentation from an XSD.

Workflow 3 — Generating test data. A consumer developer needs sample FundsXML files to test its ingestion pipeline. Hand-writing them is tedious; the **FundsXML Generator** (and FreeXmlToolkit's built-in sample-data generator) produces schema-valid documents with plausible values, ready to feed into a test suite.

Workflow 4 — Converting between FundsXML and other formats. A fund-operations analyst maintains portfolio data in Excel and needs to emit FundsXML; a data-quality analyst wants to pull FundsXML data into a spreadsheet for review. The **FundsXML CSV Converter** (and the spreadsheet conversion tools inside FreeXmlToolkit) fills this role.

Workflow 5 — Validating and signing. Schema validation, business-rule validation, and digital signatures are all available in command-line form (Chapter 10) and in GUI form (inside FreeXmlToolkit). A developer choosing between them will typically use the command-line tools for automated pipelines and the GUI tools for interactive debugging.

Table 11.1 — Tools and workflows

Workflow	Primary tool	Alternatives / overlap
Reading and exploring	FreeXmlToolkit	IDE plugins, xmllint
Schema browsing	Online Schema Viewer	FreeXmlToolkit XSD module
Test data generation	FundsXML Generator	FreeXmlToolkit sample-data generator
CSV/Excel conversion	FundsXML CSV Converter	FreeXmlToolkit spreadsheet converter
Validation	xmllint + Schematron (Chapter 10)	FreeXmlToolkit validation module
Digital signatures	FreeXmlToolkit signature module	openssl / xmlsec1

An important observation from Table 11.1: **FreeXmlToolkit appears as an alternative in almost every row**. This is not a coincidence — it is a deliberately all-in-one desktop workstation that covers most FundsXML workflows in a single installable application. Readers who want a single tool rather than a collection of them will find FreeXmlToolkit the natural home; readers who prefer specialised CLI tools, or who need a specific conversion that FreeXmlToolkit does not cover, will reach for the smaller dedicated tools alongside it.

The chapter treats FreeXmlToolkit first because of its breadth, then covers the specialised tools in the order of the TOC.

11.3 FreeXmlToolkit — in Detail

FreeXmlToolkit is an open-source desktop application, built on Java 25 and JavaFX, released under the Apache 2.0 licence. It is cross-platform (Windows, macOS, Linux), distributed both as pre-built installers and as a buildable source tree on GitHub, and authored and maintained as a community project around the FundsXML standard. Despite its general-XML naming, the toolkit has been shaped over many years of FundsXML work and contains features — a schema-aware sample-data generator, a Schematron panel wired into the XML editor, a spreadsheet converter — that are specifically useful to fund-industry users even though they also apply to any XML work.

This section covers the toolkit in five subsections: what it is structurally (11.3.1), how to install and start it (11.3.2), the main editor tabs a new user encounters (11.3.3), the workflows that matter most for FundsXML (11.3.4), and where the toolkit complements rather than replaces the command-line tools from Chapter 10 (11.3.5).

11.3.1 Structure of the Application

FreeXmlToolkit is organised around a set of **tabs** in a single main window. Each tab is dedicated to one of the application's modules; switching between tabs changes the primary view without losing the files the user has opened in other tabs. The principal tabs, in roughly the order a new user encounters them, are:

- **Welcome** — the landing page, with shortcuts to recent files, favourites, and a list of sample projects.
- **XML Editor** (the *Unified Editor*) — the main text editor for XML documents, with live XSD validation, XPath-driven IntelliSense, code folding, and a companion tree view that visualises the XML structure alongside the text. A *grid view* presents flat sections of an XML document as a table, which is handy for fund operations users who think about data in rows rather than in angle brackets.
- **XSD** — the schema browser and validator. Loads an XSD file and lets the user navigate its type hierarchy, see element definitions, and generate HTML documentation from the schema.
- **XSLT Developer** — an interactive XSLT 2.0/3.0 environment built on Saxon, with a live preview of the transformation output as the user edits the stylesheet.
- **Schematron** — a Schematron rule editor and runner, producing SVRL validation reports that are displayed inline with the source document.
- **Signature** — XML Digital Signature creation and verification.
- **FOP** — PDF generation from XML via Apache FOP, useful for turning FundsXML fact-sheet data into printable output.

- **Favourites** — a cross-tab list of frequently used XML, XSD, and Schematron files, with categories, descriptions, and one-click loading into whichever editor the user is currently in.
- **Settings** and **Help** — configuration and in-app documentation.

Not every user needs every tab. A fund-operations analyst who primarily wants to read deliveries spends most of their time in the XML Editor; a schema maintainer spends most of their time in the XSD tab; a producer pipeline developer moves between the Schematron tab and the Settings tab. The tabs are designed to be usable independently, so there is no requirement to learn every module before becoming productive in any one of them.

11.3.2 Installation and First Run

Two installation paths are supported. The **pre-built release** is the simpler one: the releases page on GitHub (<https://github.com/karlkauc/FreeXmlToolkit/releases>) provides installers for Windows, macOS, and Linux, each packaged as a zip or OS-native installer. On Windows, the installer does not require administrator rights; on macOS, the `.app` bundle is drag-dropped into `/Applications` ; on Linux, the binary is launched from the extracted directory.

The **build-from-source** path is for developers who want the latest unreleased features or who want to modify the toolkit:

```
git clone https://github.com/karlkauc/FreeXmlToolkit.git
cd FreeXmlToolkit
./gradlew run
```

Gradle downloads Java 25, JavaFX 24, and all dependencies into its local cache and builds and runs the application. The first build is slow (typically a few minutes); subsequent builds are incremental and fast.

The system requirements are modest. The toolkit runs comfortably on any machine with 4 GB of memory and a modern desktop OS, but a machine with 8 GB is more comfortable if the user plans to keep several large FundsXML files open simultaneously.

On first run, the application opens to its Welcome tab. Opening a FundsXML file for the first time is as simple as dragging the file into the main window or using the File menu. The application detects the file's format automatically — XML, XSD, XSLT, or Schematron — and routes it to the appropriate editor tab.

11.3.3 The XML Editor Up Close

The XML Editor is the tab most readers will spend most of their time in, and it deserves a slightly more detailed walkthrough.

A newly opened FundsXML document appears in the main text pane with syntax highlighting: element names, attribute names, attribute values, text content, and XML comments are each coloured distinctly, so that the user's eye can immediately parse the hierarchical structure without reading word by word. The right-hand side of the window shows a collapsible **tree view** that presents the same document as an expandable hierarchy — clicking a node in the tree view scrolls the text pane to the corresponding line, and editing the text pane updates the tree view as the user types.

Several features that distinguish the editor from a generic text editor matter for FundsXML work:

- **IntelliSense against the current XSD.** When the editor has an associated schema (either loaded explicitly through the XSD tab, or discovered automatically through `xsi:noNamespaceSchemaLocation` or `xsi:schemaLocation`), typing inside an element offers autocomplete for the valid child elements at the cursor position. This is particularly valuable inside deeply nested blocks like `RegulatoryReportings/EMT` where the field list is long and the names are easy to mistype.
- **Continuous validation.** As the user types, the editor runs XSD validation in the background and marks invalid regions of the document with red underlines, analogous to the way a modern word processor marks spelling mistakes. Hovering over the mark shows the validation error — the same messages that `xmllint` would produce (Chapter 10), but surfaced inline without the user having to run a separate command.
- **Schematron panel.** A side panel on the right accepts a Schematron rule file and runs the rules continuously against the current document, showing SVRL-style failure messages next to the editor. This brings the Chapter-10 two-stage validation model into a single interactive view: the user sees both the XSD errors and the Schematron errors as they type, rather than discovering them on the next CI run.
- **Grid view** (the *flatten* or *spreadsheet* perspective). For document sections that are essentially tabular — a portfolio of 150 positions, an EMT block of 120 fields — the grid view displays the data as a conventional spreadsheet rather than as nested XML. Editing a cell in the grid view updates the underlying XML; adding a row adds a new element. The grid view is disabled for sections whose structure is too hierarchical to fit sensibly into rows and columns.
- **Favourites.** Any file opened in any editor tab can be saved to a user-defined favourites category. The categories persist across sessions, and favourites can be re-loaded into any other editor tab (a Schematron file saved from the Schematron tab can be loaded into the XML Editor's validation panel, for example). This is an ergonomic win for users who routinely jump between the same fifteen or twenty files.

11.3.4 FundsXML-Specific Workflows

Several typical FundsXML workflows are worth describing concretely, because they show how the toolkit's generic XML features combine into task-specific productivity.

Workflow A — Reviewing an incoming delivery. A fund operations analyst receives a FundsXML file from an administrator and wants to verify that the delivery is correct before passing it on. The steps are: open the file in the XML Editor, check the inline validation markers (any red underline is a reason to stop), navigate the tree view to `ControlData` to verify sender/receiver/date, navigate to `Funds/Fund/FundDynamicData/TotalAssetValues` to cross-check the NAVs against the administrator's emailed summary, and finally load the producer's Schematron rule set into the side panel to catch business-rule violations. The whole check takes a few minutes; without the toolkit, the same check would involve several command-line invocations and a manual comparison.

Workflow B — Generating a test file. A developer working on a consumer pipeline needs a sample FundsXML file to test against. The Schema Generator module (reachable from the XSD tab) reads the FundsXML XSD and produces a complete, schema-valid sample document with plausible values (dates within a reasonable range, strings of the right length, enumerated values picked from each enumeration). The generated file is not production data, but it

exercises every structural path the consumer's parser needs to handle, and a developer can tweak the generator settings to control depth, element counts, and optional-field inclusion. This is a faster route to a test fixture than hand-writing one.

Workflow C — Building a Schematron rule set. A producer implementing the two-stage validation from Chapter 10 needs to author, test, and refine the Schematron rule file. The Schematron tab in FreeXmlToolkit combines an editor for the rules with a live run-against-the-current-XML pane: changing a rule re-runs it immediately against the loaded document, so the rule-author can iterate at the speed of typing. Once the rules are stable, the rule file is saved to disk and handed to the command-line pipeline (using the runner from Chapter 10.6.2) for production use.

Workflow D — Transforming FundsXML into a fact sheet. A distributor consumer wants to turn FundsXML data into a human-readable fact sheet. The XSLT Developer tab loads the FundsXML file on the left and an XSLT stylesheet on the right; the bottom pane shows the live-rendered output (HTML, text, or another XML format) as the user edits the stylesheet. Saxon's XSLT 2.0 and 3.0 support lets the user express sophisticated transformations — grouping positions by sector, aggregating NAVs by share class, formatting numbers with locale-aware separators — without leaving the toolkit.

11.3.5 Where FreeXmlToolkit Complements the CLI Tools

FreeXmlToolkit and the `xmllint` + Python validation pipeline from Chapter 10 are not substitutes for each other; they are complementary tools with different audiences.

The **CLI pipeline** is for production use. It runs unattended, exits with clean status codes, logs to files, and is trivially embeddable in a CI system or a nightly batch job. It is also faster: validating a 20-megabyte FundsXML file with `xmllint` takes a fraction of a second, whereas the same validation inside a JavaFX desktop application takes noticeably longer because of the UI rendering and the richer parsing setup.

The **GUI toolkit** is for interactive use. It shows errors inline with the document, lets the user navigate to them with a click, and supports the iterative edit-validate-re-edit loop that debugging a broken file requires. It also covers tasks (XSLT authoring, interactive schema browsing, sample-data generation, favourite management) that a CLI pipeline is not well-suited to.

A mature team uses both: the producer's developers and QA analysts use FreeXmlToolkit for day-to-day interactive work, the CI pipeline uses the `xmllint` + Schematron runner for automated validation before every delivery is emitted, and the same Schematron rule file is shared between the two. A rule authored inside FreeXmlToolkit runs in production inside the CI pipeline without modification, and vice versa.

11.4 The Online Schema Viewer

The Online Schema Viewer is a web-based application hosted by the FundsXML project that lets users browse the FundsXML XSD schema interactively without installing anything. It occupies a specific niche in the ecosystem: a developer who wants to answer *"what does this element mean and what are its children?"* without downloading tooling can open the viewer in a browser and have the answer within seconds.

11.4.1 What the Viewer Shows

The viewer presents the FundsXML schema as a navigable hierarchy. The entry point is usually the root element `FundsXML4`, and from there the user drills into any of its children — `ControlData`, `Funds`, `AssetMasterData`, `Documents`, `RegulatoryReportings`, and the rest — by clicking on them. Each element page shows:

- The element's declared type, with a link to the type's own definition page.
- The element's documentation annotations, drawn from the XSD's `xs:annotation/xs:documentation` elements. For FundsXML 4.2.8, many elements carry annotations in several languages (English and German are most common), and the viewer displays whichever the user's browser locale is set to.
- The element's direct children, with their own types and cardinality (`minOccurs` , `maxOccurs`).
- Attribute declarations, if any.
- Restrictions and facets for simple types (enumerations, length limits, pattern restrictions).
- Cross-references to other elements that reference or extend the current one.

11.4.2 Typical Uses

The viewer is most useful for three tasks.

Looking up a specific element. A developer writing a consumer is reading a FundsXML file and encounters `OpenClosedEnded` inside `FundStaticData`. Is this a free-text field? An enumeration? What are the allowed values? Opening the viewer, searching for `OpenClosedEnded`, and reading the page takes perhaps twenty seconds and replaces several minutes of hunting through the 39,000-line raw XSD file.

Comparing two versions of the schema. When FundsXML publishes a new minor release — 4.2.3 to 4.2.4, say — the viewer can often be pointed at either version through a version selector, and the changes between the two are listed in a changelog view. A developer upgrading a producer pipeline uses this to understand which new fields need to be populated and which deprecated fields to remove.

Exploring an unfamiliar region of the schema. A developer who has worked mostly with `Funds` and `AssetMasterData` needs to implement TPT (Chapter 8.7) and has never touched the `RegulatoryReportings/TPT` subtree before. The viewer's hyperlinked navigation makes it easy to follow the type hierarchy from the top-level `TPT` element down to the holding fields, reading documentation annotations at each step.

11.4.3 Access and Limitations

The viewer is accessed through a URL published at the official FundsXML site; Appendix E lists the current address. Access is typically free and requires no login.

Three limitations are worth knowing. First, the viewer reads the published schema, not any locally modified schema — developers working against a customised XSD need a local alternative. Second, the viewer is a *read-only* browser: it does not validate user-supplied files (for that, the user still needs `xmllint` or `FreeXmlToolkit`). Third, the viewer's update cadence lags slightly behind the XSD release cadence, so for very new minor versions the local `FundsXML4.xsd` file may be authoritative in cases where the two disagree.

11.5 FundsXML CSV Converter

A large population of fund-industry users maintains fund data in Excel or CSV files and needs to produce or consume FundsXML at the boundary. The **FundsXML CSV Converter** is the tool for that boundary. It can run in two directions: *CSV to FundsXML*, where rows in a spreadsheet are transformed into elements in an XML document, and *FundsXML to CSV*, where portions of an XML document are flattened into a table for review or editing.

The converter is available both as a standalone utility (for users who only need the conversion and do not want the full FreeXmlToolkit installation) and as a module inside FreeXmlToolkit itself (where it integrates with the rest of the toolkit's editing and validation workflows). Both versions produce equivalent results; the choice between them is a matter of preference.

11.5.1 CSV to FundsXML

The direction from CSV to XML is the more common use case. A fund operations analyst maintains a spreadsheet with columns like `ISIN`, `Name`, `Currency`, `NumberOfShares`, `NAV`, and `NavDate`, and wants to produce a FundsXML delivery from it. The converter needs two inputs:

- **The source CSV file**, with a header row naming the columns.
- **A mapping specification** that tells the converter how each column should be placed into the FundsXML document structure. The mapping is typically a small configuration file naming the target XPath for each column, plus any transformation rules (e.g., "this column is a date, and values are in DD/MM/YYYY format — convert to ISO").

Given these two inputs, the converter emits a FundsXML file that follows the XSD structure and can be fed into the Chapter-10 validation pipeline before being shipped.

The mapping specification is where the complexity lives. For a simple flat spreadsheet producing a simple ShareClass list, the mapping is short: one row per line, each row naming a column and its target XPath. For a realistic fund structure with share classes, portfolios, and dynamic data, the mapping is more involved, because the CSV's flat rows have to be folded into the hierarchical FundsXML structure. Producers who use the converter regularly typically maintain a library of mapping files, one per data type, reused across many conversion runs.

11.5.2 FundsXML to CSV

The reverse direction is useful for data review. A data-quality analyst who received a FundsXML delivery and wants to check the portfolio section against a reference list in Excel runs the converter in the FundsXML-to-CSV direction, selects the `Portfolio/Positions/Position` elements as the source, and emits a CSV with one row per position and columns for `ISIN`, `Quantity`, `MarketValue`, and whichever other fields the analyst needs. The CSV can then be opened in Excel, compared to the reference list, and the differences reviewed.

The converter does not round-trip perfectly. A FundsXML-to-CSV conversion typically loses information that does not fit into the flat tabular model — repeated elements with complex structure, language-tagged text fields, nested sub-elements. The tool marks these as "unmapped" in the output, so that the user is aware of the data that did not

make it into the CSV.

11.5.3 Comparison with FreeXmlToolkit's Grid View

The standalone converter and the FreeXmlToolkit grid view (§11.3.3) overlap in their FundsXML-to-CSV purpose, but their strengths are different. The grid view is an in-place editor: the user loads a FundsXML file, switches to grid mode for a particular section, edits cells, and saves — the underlying XML is updated in place. The CSV converter is a *file-to-file transformation*: the input is FundsXML, the output is a separate CSV, and edits to the CSV do not flow back to the XML unless the user runs the reverse conversion. Users who need to edit in place prefer the grid view; users who need a separate output file to share with an Excel user prefer the converter.

11.6 FundsXML Generator

The **FundsXML Generator** is the tool that produces sample FundsXML files — valid documents with plausible content — for use as test fixtures, development aids, and demonstration material. A consumer developer implementing a new ingestion pipeline uses the generator to create a set of test files that exercise every code path the pipeline needs to handle; a producer developer uses it to create a fresh sample file after a schema upgrade to verify that the producer's downstream handlers still work.

The generator, like the CSV converter, exists both as a standalone tool and as a feature inside FreeXmlToolkit (where it is accessible from the XSD tab's "Generate Sample" action).

11.6.1 What "Sample" Means

A sample FundsXML document is not the same as a real one. The generator aims for **structural completeness** — every optional element included at least once, every enumerated value exercised across the file, every reasonable cardinality covered — rather than for *semantic* completeness. The values are drawn from a library of plausible defaults: ISINs that look like ISINs (right length, right character classes), dates in a configurable range, country codes picked from a small set, text fields populated with identifiable placeholder strings, numeric fields with randomised values within sensible bounds.

Consumers of a sample file should know that the numbers do not represent a real fund and should not be taken at face value for business-logic testing. The file is a *structural* test vehicle: its job is to exercise the consumer's parser, not to trick it into thinking it is reading a real delivery.

11.6.2 Controlling the Generator

A sample-file generator has several knobs that a developer typically needs to control:

- **Depth**: how deeply to recurse into optional substructures. Setting the depth to 1 produces a minimal file (only the required elements); setting it higher progressively includes more optional content.
- **Array sizes**: how many instances to produce for elements with `maxOccurs > 1`. A portfolio with a hundred positions exercises different code paths from a portfolio with two positions; both are useful for different tests.

- **Optional-field inclusion:** whether to include every optional element or only a subset. Some consumers fail gracefully when optional elements are absent; others fail gracefully when they are present. A comprehensive test suite exercises both cases.
- **Random-seed fixing:** whether the random values are generated from a fixed seed (reproducible runs) or from system entropy (different values each time). For automated tests, a fixed seed is essential; for ad-hoc exploration, random values are fine.
- **Target schema version:** which FundsXML version to target. A consumer that supports both 4.2.3 and 4.2.8 should be tested with fixtures from both versions.

Production usage of the generator typically involves a short configuration file that sets these options, plus a script that runs the generator on every CI build to produce a fresh set of fixtures that the consumer pipeline can then run against.

11.6.3 Relationship to Real Test Data

The generator is useful but not a substitute for real test data. Once a producer pipeline is running, the best regression fixtures are captured snapshots of actual deliveries — anonymised if necessary — because they exercise the quirks and edge cases that a generator does not anticipate. The mature practice is to use generator output for initial development and schema-upgrade testing, and captured real data for ongoing regression testing. Chapter 13 returns to this distinction in the context of implementation projects.

11.7 IDE Integration — IntelliJ, VS Code, Eclipse

Many FundsXML developers — both on the producer side and on the consumer side — spend most of their working day inside an integrated development environment rather than in a dedicated XML tool. For those developers, the most valuable integration is *inside the IDE they already use*, with FundsXML validation and navigation available as first-class features of the editing experience. All three major cross-platform Java and polyglot IDEs — IntelliJ IDEA, Visual Studio Code, and Eclipse — support FundsXML well through their generic XML facilities. No FundsXML-specific plugin is required in any of the three; what is required is the standard XML support plus a correctly configured schema location.

11.7.1 IntelliJ IDEA

IntelliJ IDEA's bundled XML support is comprehensive and works against any XSD out of the box. Opening a FundsXML file in IntelliJ gives the user syntax highlighting, fold markers, and navigation; enabling schema validation requires telling IntelliJ where to find `FundsXML4.xsd`.

The simplest path is the `xsi:noNamespaceSchemaLocation` attribute on the root element, which IntelliJ respects. If the FundsXML file is stored in the same directory as `FundsXML4.xsd` (or in a directory referenced by a relative path in the attribute), IntelliJ finds the schema automatically and enables validation, autocomplete, and "go to definition" on every element. The error markers in the gutter match the `xmlint` errors from Chapter 10.

For projects that store the schema separately from the data, IntelliJ's **Settings** → **Languages & Frameworks** → **Schemas and DTDs** lets the user map a schema file to a pattern of XML files. Mapping `FundsXML4.xsd` to `*.xml` in a FundsXML-data directory enables full schema support without needing the `xsi:` attributes.

IntelliJ's XPath evaluator (under the **Edit** → **Find** → **Find Usages / XPath Search** menu) is a useful companion for exploring a FundsXML file by query. The XSLT debugger, while not FundsXML-specific, is equally useful for developing XSLT stylesheets that transform FundsXML into other formats.

11.7.2 Visual Studio Code

Visual Studio Code does not ship with XML support by default, but the **Red Hat XML extension** (`redhat.vscode-xml`) is free, widely used, and provides a comparable level of functionality to IntelliJ's built-in support. Installing it from the VS Code marketplace takes under a minute; once installed, it detects `.xml` files automatically and offers schema-driven features.

Schema association works the same way as in IntelliJ: either the `xsi:noNamespaceSchemaLocation` attribute on the root element, or a project-level configuration in VS Code's settings that maps schemas to file patterns. The Red Hat extension reads a `.settings/org.eclipse.wst.xml.core.prefs` file (inherited from Eclipse WTP conventions) and also a VS Code-native configuration, so either approach works.

The extension provides continuous validation, auto-formatting, XPath evaluation, and XSLT 3.0 support through its bundled Saxon-HE. For a developer who lives in VS Code, it is the most capable path into FundsXML work, and the experience closely mirrors FreeXmlToolkit's XML Editor without requiring a separate desktop application to be running.

11.7.3 Eclipse

Eclipse has supported XSD-aware XML editing for many years through the **Eclipse Web Tools Platform (WTP)**, which is included in every Eclipse IDE for Java EE Developers and most other Eclipse packages. The WTP XML editor is mature, well-tested, and fully schema-aware.

For FundsXML work in Eclipse, the typical configuration is to register `FundsXML4.xsd` in **Preferences** → **XML** → **XML Catalog**, which maps a public or system identifier to the local schema file. Once registered, any FundsXML file opened in the editor resolves its schema automatically and gets validation, autocomplete, and "open declaration" navigation.

Eclipse's support for XSLT and Schematron comes through additional plugins (the Eclipse Marketplace has several; the names change over time, so the current recommendation is to search the marketplace for "Schematron" and "XSLT" and pick the most recently updated result). The core XML editor is what most FundsXML developers use day to day; the specialised plugins are reached for only when a specific workflow needs them.

11.7.4 Which IDE to Choose

All three IDEs provide equivalent FundsXML functionality for the core editing tasks: syntax highlighting, schema validation, autocomplete, and navigation. The choice between them is almost always determined by the surrounding toolchain rather than by any FundsXML-specific feature. A developer whose project is Java and uses Gradle or Maven will find IntelliJ or Eclipse natural; a developer whose project is polyglot (Python, Go, Rust, JavaScript, with

XML at the boundary) will find VS Code lighter-weight and faster to start. None of the three has a compelling advantage for FundsXML work specifically, and a team with developers using all three can share Schematron files, XSD files, and code without friction.

11.8 Common Pitfalls

- **Using a desktop tool for a pipeline task.** FreeXmlToolkit is excellent for interactive work but is not designed for unattended batch execution. A pipeline that invokes the toolkit in headless mode to validate deliveries is slower, more fragile, and harder to debug than the `xmllint` + Schematron pipeline from Chapter 10. Use the right tool for the right job.
- **Using CLI tools for exploration.** The complementary mistake: trying to understand an unfamiliar FundsXML file by running `xmllint --xpath` queries from the command line one at a time. It is possible but tedious; an interactive editor with tree view and XPath evaluation is dramatically faster for exploration tasks.
- **Running the Online Schema Viewer on a schema version that differs from the production schema.** If the producer is using a slightly modified or slightly older schema than the one the viewer shows, some element definitions differ in ways that matter. For authoritative answers about *your* schema, use FreeXmlToolkit's XSD tab or a local copy of the viewer, not the public one.
- **Forgetting to update mapping files when the schema changes.** The CSV Converter relies on mapping files that describe how to map columns to XPaths. A FundsXML minor release that adds new fields or renames existing ones will not break the converter until the next run, and even then the breakage may be silent (new fields simply not populated). Include a test step in the pipeline that re-runs the converter after every schema update and diffs the output against an expected fixture.
- **Installing a generator with random values and forgetting to fix the seed.** A consumer test suite that runs against generator output *without* a fixed seed produces a different set of test data on every run, and failures become non-reproducible. Always fix the seed for reproducible tests; randomise only for ad-hoc exploration.
- **Treating the IDE XML validation as equivalent to CI XML validation.** Different XSD validators can produce subtly different results, particularly around edge cases like default attribute values or identity-constraint handling. A file that validates in IntelliJ should still be run through `xmllint` (or whichever validator the pipeline uses) before emission, because the pipeline's validator is the authoritative one.
- **Relying on a single tool.** A team that depends entirely on FreeXmlToolkit is vulnerable to the day FreeXmlToolkit has an unresolved bug; a team that depends entirely on the Online Schema Viewer is vulnerable to network outages. Mature teams keep at least two tools available for any critical workflow, so that an outage of one does not stop work entirely.

11.9 Key Takeaways

- The FundsXML tool ecosystem serves a handful of distinct workflows — reading and exploring, schema browsing, test data generation, format conversion, validation, signing. Most of the tools in the ecosystem serve one or more of these workflows, and many of them overlap.

- **FreeXmlToolkit** is the multi-function desktop workstation that covers almost every workflow in a single application: XML editing with inline XSD and Schematron validation, schema browsing, sample-data generation, XSLT development, XML-to-PDF generation, digital signatures, and a favourites system for cross-tab file management. It is the default recommendation for interactive FundsXML work.
- **The Online Schema Viewer** is a zero-install browser-based way to look up element definitions, documentation, and type hierarchies in the FundsXML XSD. It is the fastest path to answering "what does this element mean?" without launching a desktop tool.
- **The FundsXML CSV Converter** bridges between spreadsheets and FundsXML in both directions, and is used by producers whose data sources are Excel-based and by consumers who want to review FundsXML data in a tabular form.
- **The FundsXML Generator** produces schema-valid sample files for testing and development. Its output is structurally comprehensive but not semantically real; use it for pipeline exercises, not for business-logic testing.
- **IDE integration** — IntelliJ IDEA, Visual Studio Code (with the Red Hat XML extension), and Eclipse (with WTP) — is the preferred path for developers who already live in an IDE. All three support FundsXML through generic XSD-aware XML editing once the schema is configured, and no FundsXML-specific plugin is required.
- The command-line tools from Chapter 10 (`xmllint` , `lxml.isoschematron`) and the interactive tools from this chapter are complementary, not competitive. A mature team uses both: CLI for pipelines, GUI for interactive debugging and development.

With the tooling landscape mapped, the next question is: *how does a FundsXML pipeline fit into a larger system landscape?* Chapter 12 treats that question. It covers typical architecture scenarios, the choice between programming languages for FundsXML work, strategies for reading and writing at scale, database and data-warehouse integration, and the scheduling and automation patterns that take a FundsXML producer from a proof-of-concept to a production pipeline.



FundsXML in the System Landscape

Integration into existing IT architectures

12.1 Setting the Scene: From File to System

Chapters 10 and 11 treated FundsXML as a file on disk — a thing you validate, open in an editor, transform with XSLT. Real production systems rarely treat it that way. In a production architecture, a FundsXML file is the output of a pipeline that read from a database, assembled content from several source systems, and will shortly be emitted over an SFTP or HTTPS channel to a distributor. Or it is the input of a pipeline that fetched the file from a drop-box, parsed it, decomposed it into table rows, and loaded those rows into a downstream warehouse. The file is a transport format; the architecture around it is the interesting part.

This chapter is about that architecture. It describes the typical system scenarios that asset managers and distributors deploy, compares the four programming languages that dominate FundsXML implementation work (Java, Python, C#, JavaScript), walks through the strategies for reading and processing XML at scale (DOM, SAX, StAX, XPath), treats database integration and data-warehousing patterns, and closes with the automation and scheduling patterns that turn a working pipeline into a production pipeline.

A disclosure on code execution in this chapter: three of the four language examples have been **run in the environment where this chapter was written**, against a minimal but schema-valid FundsXML sample file, and the output shown in the chapter is the real output of those runs. The Python example uses `lxml`; the Node.js example uses `fast-xml-parser`; the Java example uses JAXP with XPath. The fourth example, the C# listing using LINQ to XML, follows standard idiomatic practice but was **not executed** in this environment because the .NET runtime was not installed. Readers who want to verify the C# example should copy it into a .NET project and run it themselves; the listing is structurally complete and should run without modification.

By the end of this chapter, you should be able to:

- name the handful of architectural patterns that recur in FundsXML production deployments;
- choose a programming language and XML library for a new FundsXML implementation based on the existing stack rather than on theoretical preferences;
- write a minimal reader for a FundsXML file in any of Java, Python, C#, or JavaScript;
- choose between DOM-based, streaming (SAX/StAX), and XPath-based reading strategies for different file sizes and access patterns;
- decide whether to store FundsXML data in a relational database, a document database, or an XML-native database;
- design an ETL pipeline that loads FundsXML deliveries into a data warehouse;
- select an automation and scheduling approach that matches the delivery frequency and the reliability requirements of the pipeline.

12.2 Typical Architecture Scenarios

Before we look at any code, a map of the architecture patterns that recur in real FundsXML deployments. Most production systems resemble one of the following scenarios, or a composition of two or three of them.

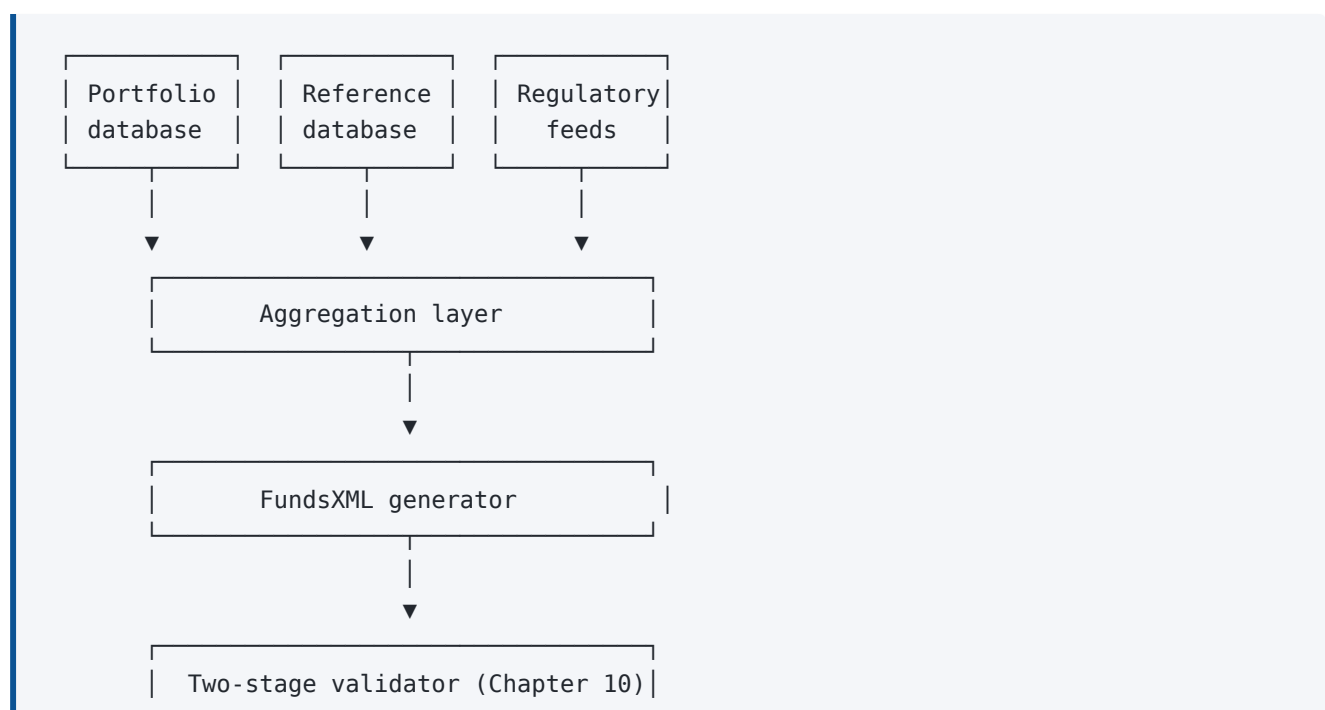
12.2.1 The Producer Pipeline

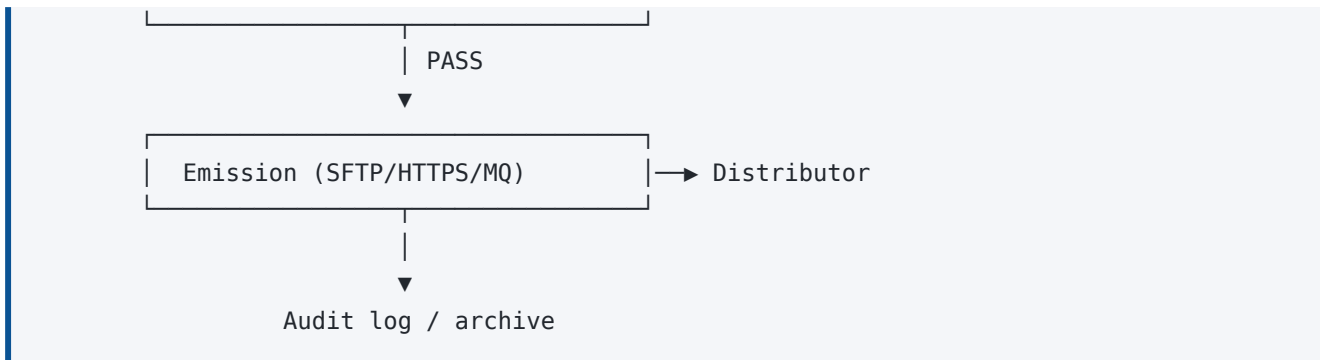
The producer pipeline is the sender of FundsXML data. Its inputs are internal databases, spreadsheets, and upstream feeds; its output is a FundsXML file (or a stream of files) that is shipped to one or more consumers. A typical producer pipeline has the following components:

- An **aggregation layer** that queries internal databases and collects the fund reference data, the portfolio positions, the regulatory template inputs, and everything else the delivery needs to carry.
- A **FundsXML generator** that takes the aggregated data and produces an XML document against the target schema version.
- A **validation layer** that runs the Chapter-10 two-stage validation (XSD + Schematron) and either passes the file to emission or rejects it with a diagnostic.
- An **emission layer** that ships the validated file to its destination over SFTP, HTTPS, or a message queue.
- An **audit log** that records every delivery, its status, and any validation failures for later investigation.

The Europa Growth Fund's monthly delivery goes through exactly this pipeline: the fund administrator's systems aggregate the portfolio data around 20:00 CET on the last business day of the month, the generator produces a FundsXML file around 21:30, the validator checks it around 21:35, and the emission layer hands it off to the delivery channels by 22:00 to the distribution countries' drop-boxes.

Figure 12.1 — Producer pipeline





12.2.2 The Consumer Pipeline

The consumer pipeline is the mirror image. Its inputs are FundsXML files arriving from upstream producers; its outputs are rows in a database, entries in a reporting warehouse, rendered fact sheets, or messages on a downstream bus. A typical consumer pipeline has:

- An **ingestion layer** that fetches incoming files from a drop-box, queue, or HTTPS endpoint and moves them into a staging area.
- A **validation layer** that runs the same two-stage validation as the producer did. A consumer that trusts the producer's validation completely is a consumer that eventually gets hurt by a producer bug; running the validation on the consumer side is the standard defensive practice.
- A **parsing and decomposition layer** that reads the FundsXML file and extracts the data the consumer's downstream systems care about. A distributor cares about fund identities, NAVs, and costs; a data warehouse cares about portfolio positions; a regulator cares about [RegulatoryReportings](#) modules. Different consumers decompose differently.
- A **loading layer** that writes the decomposed data into the downstream storage, handling idempotency (so that a re-delivered file does not produce duplicate rows), versioning (so that an AMEND delivery replaces the earlier values cleanly), and error recovery.
- An **audit log** on the consumer side as well, recording every file received and the outcome of its processing.

12.2.3 The Distributor Dispatcher

A variant of the consumer pipeline is the **dispatcher**: a consumer that receives many FundsXML files and routes each to a different downstream system based on its content. A large retail distributor might have separate systems for retail KID disclosure, for institutional reporting, for internal fund screening, for the trading desk, and so on; each of those systems needs a subset of the data in each FundsXML file. The dispatcher reads the `ControlData`, `RegulatoryReportings` block, and fund identifiers, then routes the file (or a transformed projection of it) to the right subset of downstream systems. The Chapter-4 description of the dispatcher's first five tasks — recognise, authenticate, route, deduplicate, sequence — applies exactly here.

12.2.4 The Data Warehouse Loader

A different variant is the **warehouse loader**: a consumer that is not interested in the ongoing operational flow but in the historical record. Every FundsXML file received is shredded into relational rows and loaded into a data warehouse alongside years of accumulated history, so that analysts can query fund performance, portfolio composition

changes, and regulatory disclosure trends over long periods. The warehouse loader optimises for *bulk insert* rather than for *real-time response*; it accepts a latency of hours between file receipt and queryability in exchange for being able to handle hundreds of files per day. §12.6 treats the ETL patterns these loaders use.

12.2.5 The Mixed-Workflow Asset Manager

A mid-sized asset manager typically runs **all four scenarios at once**: a producer pipeline for outgoing FundsXML deliveries to distributors; a consumer pipeline for incoming deliveries from fund administrators; a dispatcher for routing incoming regulatory reports to different internal groups; a warehouse loader for historical analytics. The pipelines share some infrastructure (the same validation library, the same schema files, the same audit logging system) but are structurally distinct. Keeping the responsibilities separate is the way mature teams manage complexity.

12.3 Programming Language Comparison

The FundsXML ecosystem is language-agnostic in the sense that any language with a competent XML library can produce and consume FundsXML. In practice, four languages dominate: **Java**, **Python**, **C#**, and **JavaScript** (specifically Node.js, for server-side code). This section walks through each one with a minimal but complete working example: open a FundsXML file, extract the fund's LEI and its total NAV, and print them. Side by side, the four examples show how idiomatic reading looks in each language.

The task is deliberately simple. A production pipeline does much more — multi-file handling, error recovery, logging, validation, and so on — but the minimal task shows how the language and its XML library feel. Readers picking a language for a new project should run each example, mentally scale it up to production, and pick the one that fits their existing stack most comfortably.

12.3.1 The Common Test File

All four examples read the same FundsXML file — a minimal but schema-valid document containing one fund with a LEI, a name, a currency, and a total net asset value.

```
<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-LANG-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
```



```

</DataSupplier>
<DataOperation>INITIAL</DataOperation>
<Language>en</Language>
</ControlData>
<Funds>
  <Fund>
    <Identifiers><LEI>549300ABCDEFGHJIJ1234</LEI></Identifiers>
    <Names><OfficialName>Europa Growth Fund</OfficialName></Names>
    <Currency>EUR</Currency>
    <SingleFundFlag>true</SingleFundFlag>
    <FundDynamicData>
      <TotalAssetValues>
        <TotalAssetValue>
          <NavDate>2026-03-31</NavDate>
          <TotalAssetNature>OFFICIAL</TotalAssetNature>
          <TotalNetAssetValue>
            <Amount ccy="EUR">464552848.78</Amount>
          </TotalNetAssetValue>
        </TotalAssetValue>
      </TotalAssetValues>
    </FundDynamicData>
  </Fund>
</Funds>
</FundsXML4>

```

This file validates against `FundsXML4.xsd` with the `xmllint` command from Chapter 10. The four language examples below all produce the same output:

```

Europa Growth Fund
LEI:  549300ABCDEFGHJIJ1234
NAV:  464552848.78 EUR

```

12.3.2 Python with lxml

Python's `lxml` library is the de facto standard for XML work in the Python ecosystem. It is built on top of libxml2 (the same C library that `xmllint` uses) and combines excellent performance with a Pythonic API. For FundsXML work, `lxml` offers DOM-style navigation through `ElementTree`, XPath queries, and schema validation all in one package.

```

#!/usr/bin/env python3
"""Read a FundsXML file and print the fund LEI and total NAV."""
import sys
from lxml import etree

def read_fund(path: str) -> None:
    doc = etree.parse(path)
    fund = doc.find("./Funds/Fund")
    if fund is None:

```

```

    print(f"{path}: no Fund element found", file=sys.stderr)
    sys.exit(1)

    lei = fund.findtext("Identifiers/LEI") or "(no LEI)"
    name = fund.findtext("Names/OfficialName") or "(unnamed)"

    tav = fund.find(
        "FundDynamicData/TotalAssetValues/TotalAssetValue/"
        "TotalNetAssetValue/Amount"
    )
    if tav is None:
        print(f"{name} ({lei}): no TotalNetAssetValue found")
        return

    amount = tav.text
    currency = tav.get("ccy")
    print(f"{name}")
    print(f"  LEI: {lei}")
    print(f"  NAV: {amount} {currency}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("usage: read_fund.py <file.xml>", file=sys.stderr)
        sys.exit(2)
    read_fund(sys.argv[1])

```

Run with `python3 read_fund.py egf-sample.xml`. The API is compact, error handling is straightforward, and the code reads top-to-bottom without ceremony. For production use, `lxml` also offers:

- **Schema validation** via `etree.XMLSchema` (equivalent to running `xmllint --schema`), integrated directly into the parsing step so that invalid documents fail on load.
- **Schematron validation** via `lxml.isoschematron` (which we used in Chapter 10).
- **Incremental parsing** via `etree.iterparse`, for files too large to load into memory.
- **XPath 1.0** queries via the `xpath()` method, for more complex extraction tasks.

Python's strengths for FundsXML work are the speed of initial development, the range of adjacent libraries (pandas for data-frame manipulation, SQLAlchemy for database work, Airflow for orchestration), and the low cognitive overhead of small scripts. Its weaknesses are memory use on very large files (DOM-based parsing keeps the whole document in memory) and the GIL's limits on in-process parallelism.

12.3.3 Java with JAXP and XPath

Java is the most widely used language for FundsXML work on the producer side, primarily because it is the dominant language of enterprise asset-management systems. The JAXP API (Java API for XML Processing) ships with the JDK and provides both DOM and SAX parsing; `javax.xml.xpath.XPathFactory` provides XPath 1.0 queries against a parsed DOM.

```
// ReadFund.java – Read a FundsXML file with JAXP + XPath and print LEI + NAV.
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class ReadFund {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("usage: java ReadFund <file.xml>");
            System.exit(2);
        }

        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        dbf.setNamespaceAware(false);
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File(args[0]));

        XPath xpath = XPathFactory.newInstance().newXPath();
        Node fund = (Node) xpath.evaluate("/FundsXML4/Funds/Fund", doc,
            javax.xml.xpath.XPathConstants.NODE);
        if (fund == null) {
            System.err.println(args[0] + ": no Fund element found");
            System.exit(1);
        }

        String lei = xpath.evaluate("Identifiers/LEI", fund);
        String name = xpath.evaluate("Names/OfficialName", fund);
        Node amt = (Node) xpath.evaluate(
            "FundDynamicData/TotalAssetValues/TotalAssetValue/"
            + "TotalNetAssetValue/Amount", fund,
            javax.xml.xpath.XPathConstants.NODE);

        if (amt == null) {
            System.out.println(name + " (" + lei + "): no TotalNetAssetValue");
            return;
        }

        String amount = amt.getTextContent();
        String currency = amt.getAttributes().getNamedItem("ccy").getNodeValue();

        System.out.println(name);
        System.out.println("  LEI:  " + lei);
        System.out.println("  NAV:  " + amount + " " + currency);
    }
}
```

Compile and run with `javac ReadFund.java && java ReadFund egf-sample.xml`. The code is several times longer than the Python version and noticeably more ceremonious: the `DocumentBuilderFactory` / `DocumentBuilder` / `Document` triad, the explicit `(Node)` cast after each XPath evaluation, the uppercase qualified constant `javax.xml.xpath.XPathConstants.NODE`. Java's strength is not brevity; it is robustness, static type checking, and the maturity of its libraries.

Everything in the example above uses APIs that ship with the JDK itself — `javax.xml.parsers.*`, `javax.xml.xpath.*`, and the `org.w3c.dom.*` interfaces — so no third-party dependency is required. JAXP also covers two other styles of XML processing that are useful for different tasks, and both remain inside the JDK:

- **StAX** (`javax.xml.stream.XMLStreamReader`) is the JDK's streaming parser, pulled rather than pushed. It is the right tool when the document is too large to hold in memory as a DOM — a multi-gigabyte TPT delivery covering hundreds of funds, for example — because StAX reads one event at a time and lets the application keep only the state it needs. The code is more involved than the DOM-plus-XPath style shown above, but the memory footprint is essentially constant.
- **SAX** (`org.xml.sax.*`) is the older event-driven parser, still present in the JDK for compatibility. Most new code that would have used SAX ten years ago now uses StAX instead.

Schema validation is also built in: `javax.xml.validation.SchemaFactory` loads `FundsXML4.xsd` into a `Schema` object, which can then be attached to either the `DocumentBuilderFactory` (for validate-on-parse) or used directly through a `Validator` against a pre-parsed source. The approach is the exact JDK equivalent of the `xmllint --schema` invocations we used throughout Chapters 5–8.

Note that this section deliberately stays inside the JDK's own XML stack. Third-party Java libraries for XML exist — higher-level DOM alternatives such as DOM4J and JDOM, the Woodstox StAX implementation, and the Jakarta XML Binding (JAXB) framework that generates typed Java classes from an XSD — and some production Java projects do use them. The examples in this chapter stay with JAXP because it is zero-dependency, ships with every JDK, and is enough to read and write FundsXML documents in a robust and production-grade way.

Java's strengths for FundsXML are performance at scale, mature tooling (FreeXmlToolkit itself is built in Java), static type safety, and the enterprise-grade support stacks (Spring, Java EE) that most asset managers already run. Its weaknesses are verbosity and the startup cost of the JVM, which makes Java a poor choice for short ad-hoc scripts.

12.3.4 C# with LINQ to XML

C# and .NET dominate FundsXML work at insurance companies and at a subset of Microsoft-shop asset managers. The idiomatic XML API in modern C# is **LINQ to XML**, which provides a fluent query-style interface built on `XDocument`, `XElement`, and `XAttribute`. LINQ to XML is part of `System.Xml.Linq` in the base class library and does not require any additional package.

```
// ReadFund.cs – Read a FundsXML file with LINQ to XML and print LEI + NAV.
// Build: dotnet run --project ReadFund.csproj <file.xml>
using System;
using System.Xml.Linq;

class ReadFund {
    static int Main(string[] args) {
```

```

        if (args.Length != 1) {
            Console.Error.WriteLine("usage: ReadFund <file.xml>");
            return 2;
        }

        var doc = XDocument.Load(args[0]);
        var fund = doc.Root?.Element("Funds")?.Element("Fund");
        if (fund == null) {
            Console.Error.WriteLine($"{args[0]}: no Fund element found");
            return 1;
        }

        string lei = (string?)fund.Element("Identifiers")?.Element("LEI") ?? "(no
LEI)";
        string name = (string?)fund.Element("Names")?.Element("OfficialName") ?? "(
unnamed)";

        var amount = fund
            .Element("FundDynamicData")?
            .Element("TotalAssetValues")?
            .Element("TotalAssetValue")?
            .Element("TotalNetAssetValue")?
            .Element("Amount");

        if (amount == null) {
            Console.WriteLine($"{name} ({lei}): no TotalNetAssetValue found");
            return 0;
        }

        string value = amount.Value;
        string currency = (string?)amount.Attribute("ccy") ?? "";

        Console.WriteLine(name);
        Console.WriteLine($"  LEI:  {lei}");
        Console.WriteLine($"  NAV:  {value} {currency}");
        return 0;
    }
}

```

The C# version sits stylistically between Python and Java: more concise than Java, slightly more ceremonious than Python. The `?.` null-conditional operator and the `(string?)` cast on `XElement` are the modern C# idioms for safe navigation through an XML tree. The fluent `.Element(...).Element(...).Element(...)` chain is the LINQ to XML signature.

(Note: this listing was not executed in the environment where this chapter was written, because the .NET runtime was not installed. The code is structurally complete and follows standard idiomatic practice; readers who want to verify it should install the .NET SDK and run `dotnet run` in a new project containing this file.)

For production C# work, additional options include:

- **XmlSerializer** for class-based deserialisation (the C# equivalent of JAXB).

- **XmlReader** for streaming access to very large files.
- **XPath** via `XPathSelectElement` / `XPathSelectElements` extension methods, for XPath-based queries instead of the fluent `.Element(...)` chain.

C#'s strengths are the tight integration with Microsoft-shop tools (SQL Server, Azure, SharePoint), strong static typing, and modern language ergonomics (records, pattern matching, nullable reference types). Its weaknesses are the cross-platform story (better than it used to be but still slightly less smooth than Java or Python) and the narrower open-source ecosystem around XML-specific tooling.

12.3.5 JavaScript (Node.js) with fast-xml-parser

Node.js dominates FundsXML work on the distributor side of the house, where many retail distribution platforms are built on JavaScript and TypeScript. XML is not a native JavaScript strength, but the **fast-xml-parser** npm package provides a competent pure-JavaScript parser that turns an XML document into a plain JavaScript object tree for navigation.

```
// read_fund.mjs – Read a FundsXML file with fast-xml-parser and print LEI + NAV.
import { readFileSync } from "node:fs";
import { XMLParser } from "fast-xml-parser";

function readFund(path) {
  const xml = readFileSync(path, "utf8");
  const parser = new XMLParser({
    ignoreAttributes: false,
    attributeNamePrefix: "@_",
  });
  const doc = parser.parse(xml);

  const fund = doc?.FundsXML4?.Funds?.Fund;
  if (!fund) {
    console.error(`${path}: no Fund element found`);
    process.exit(1);
  }

  const lei = fund.Identifiers?.LEI ?? "(no LEI)";
  const name = fund.Names?.OfficialName ?? "(unnamed)";
  const tav =
    fund.FundDynamicData?.TotalAssetValues?.TotalAssetValue
      ?.TotalNetAssetValue?.Amount;

  if (!tav) {
    console.log(`${name} (${lei}): no TotalNetAssetValue found`);
    return;
  }

  const amount = typeof tav === "object" ? tav["#text"] : tav;
  const currency = typeof tav === "object" ? tav["@_ccy"] : "";

  console.log(name);
```

```

    console.log(` LEI:  ${lei}`);
    console.log(` NAV:  ${amount} ${currency}`);
  }

  const path = process.argv[2];
  if (!path) {
    console.error("usage: node read_fund.mjs <file.xml>");
    process.exit(2);
  }
  readFund(path);

```

Install the parser with `npm install fast-xml-parser`, then run `node read_fund.mjs egf-sample.xml`. The JavaScript version is comparable in length to Python. The main idiomatic difference is that `fast-xml-parser` turns the XML into a plain JavaScript object, which means navigation is through normal object-property access (`doc.FundsXML4.Funds.Fund.Identifiers.LEI`) rather than through an XML-specific API. Attributes are surfaced as specially-named properties (`@_ccy`, following the parser's default convention) and elements with mixed content expose a `#text` property for the text value.

For production JavaScript work, alternatives include:

- **sax** for SAX-style streaming access to large files.
- **xml2js** as a simpler alternative to `fast-xml-parser`, with a slightly slower but more configurable API.
- **libxmljs2** as a native-binding wrapper around `libxml2` for developers who prefer a lower-level API and are willing to accept the native-binding complexity.
- The browser-native **DOMParser** and **XPath evaluator** APIs, for client-side FundsXML work in a browser (rare, but occasionally useful for internal dashboards).

JavaScript's strengths are the ubiquity of the runtime, the speed of development, and the natural fit with modern web-facing distributor architectures. Its weaknesses are XML-specific library maturity (JavaScript XML libraries are generally less mature than their Java, Python, or C# counterparts) and the lack of a native XSD validator in the standard libraries — for schema validation, a Node.js pipeline typically shells out to `xmllint` or uses a `libxml2` binding.

12.3.6 Side-by-Side Comparison

Table 12.1 — Programming language comparison for FundsXML

Aspect	Java	Python	C#	JavaScript/Node
Primary library	JAXP (JDK)	lxml	System.Xml.Linq	fast-xml-parser
Lines of minimal reader	~35	~25	~30	~30
Schema validation	Built-in (JAXP)	Built-in (lxml)	Built-in (XmlSchema)	External (xmllint)
Schematron	External (ph-schematron)	Built-in (lxml)	External (Saxon)	External (lxml shellout)
Streaming parser	StAX	iterparse	XmlReader	sax
Typical deployment	Enterprise producer	Scripting + ETL	Microsoft-shop producer	Distributor dispatcher
Startup time	Slow (JVM)	Fast	Medium	Fast
Memory footprint (DOM)	High	Medium	Medium	Medium
Developer productivity	Medium	High	Medium	High

The practical recommendation is simple: use the language your team already uses. All four languages can produce and consume FundsXML competently; none of them has a compelling FundsXML-specific advantage that outweighs the cost of adopting a new language. A Java shop should use Java; a Python-shop data engineering team should use Python; a Microsoft-shop fund administrator should use C#; a distributor running a modern retail platform on Node.js should use Node.js. The minor differences between them become irrelevant compared to the productivity cost of maintaining code in a language the team is not fluent in.

One further approach — XSLT — does not fit neatly into the table above because it is a transformation language rather than a general-purpose programming language, but it is common enough in FundsXML practice to deserve its own treatment. §12.3.7 covers it next.

12.3.7 XSLT — A Different Kind of Transformation

The four languages compared above are all *imperative*: the programme tells the computer, step by step, how to open the file, walk the tree, and extract the values. XSLT (Extensible Stylesheet Language Transformations) works the other way round: a stylesheet declares *patterns* that match elements in the input tree, and the XSLT processor walks the tree and applies whichever pattern matches at each node. The result is a new document — HTML, CSV, plain text, or another XML shape. XSLT is a W3C standard, ships with virtually every XML toolkit, and is the natural tool when the task is "transform a FundsXML document into another format" rather than "embed XML reading into a larger application". Appendix B §B.3 provides a quick reference for the language itself; this section shows two runnable examples.

Example 1 — HTML fact sheet. The task: produce a one-page HTML summary of the Europa Growth Fund from the standard FundsXML delivery file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" indent="yes" encoding="UTF-8"/>

  <xsl:template match="/">
    <xsl:variable name="fund" select="/FundsXML4/Funds/Fund"/>
    <xsl:variable name="nav"
      select="$fund/FundDynamicData/TotalAssetValues/TotalAssetValue
        /TotalNetAssetValue/Amount"/>
    <html>
      <head><title><xsl:value-of select="$fund/Names/OfficialName"/></title></head>
      <body>
        <h1><xsl:value-of select="$fund/Names/OfficialName"/></h1>
        <table border="1" cellpadding="4">
          <tr><td>LEI</td>
            <td><xsl:value-of select="$fund/Identifiers/LEI"/></td></tr>
          <tr><td>Currency</td>
            <td><xsl:value-of select="$fund/Currency"/></td></tr>
          <tr><td>NAV</td>
            <td><xsl:value-of select="$nav"/>
              <xsl:text> </xsl:text>
              <xsl:value-of select="$nav/@ccy"/></td></tr>
          <tr><td>NAV Date</td>
            <td><xsl:value-of select="$fund/FundDynamicData
              /TotalAssetValues/TotalAssetValue/NavDate"/></td></tr>
          <tr><td>Content Date</td>
            <td><xsl:value-of
              select="/FundsXML4/ControlData/ContentDate"/></td></tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Run with `xsltproc factsheet.xsl egf-sample.xml > factsheet.html` (or with Saxon: `java -jar saxon-he.jar -s:egf-sample.xml -xsl:factsheet.xsl -o:factsheet.html`). The output is a self-contained HTML page:

```
<html>
  <head><title>Europa Growth Fund</title></head>
  <body>
    <h1>Europa Growth Fund</h1>
    <table border="1" cellpadding="4">
      <tr><td>LEI</td><td>529900T8BM49AURSD055</td></tr>
      <tr><td>Currency</td><td>EUR</td></tr>
      <tr><td>NAV</td><td>248537281.44 EUR</td></tr>
```

```

        <tr><td>NAV Date</td><td>2026-03-31</td></tr>
        <tr><td>Content Date</td><td>2026-03-31</td></tr>
    </table>
</body>
</html>

```

The entire logic is in a single `<xsl:template match="/">` that fires at the document root and emits HTML by pulling values from the FundsXML tree via XPath expressions. The same XPath paths that appeared in the Java and Python examples reappear here — `$fund/Identifiers/LEI`, `$fund/Names/OfficialName`, the deep path down to `TotalNetAssetValue/Amount` — but the surrounding plumbing is radically shorter because XSLT's output model handles the HTML serialisation directly.

Example 2 — CSV of portfolio positions. The task: export the fund's assets as a CSV file with one row per position, suitable for loading into a spreadsheet or a downstream ETL pipeline. The stylesheet uses an `xsl:key` to join the `AssetMasterData/Asset` entries (which carry the ISIN and name) with the `Position` entries in the portfolio (which carry the market value).

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" encoding="UTF-8"/>
  <xsl:strip-space elements="*" />

  <xsl:key name="pos" match="//Position/*" use="UniqueID" />

  <xsl:template match="/">
    <xsl:text>ISIN,Name,Currency,MarketValue&#10;</xsl:text>
    <xsl:for-each select="/FundsXML4/AssetMasterData/Asset">
      <xsl:variable name="pos" select="key('pos', UniqueID)" />
      <xsl:value-of select="Identifiers/ISIN" />
      <xsl:text>,</xsl:text>
      <xsl:value-of select="AssetName" />
      <xsl:text>,</xsl:text>
      <xsl:value-of select="$pos/TotalValue/Amount/@ccy" />
      <xsl:text>,</xsl:text>
      <xsl:value-of select="$pos/TotalValue/Amount" />
      <xsl:text>&#10;</xsl:text>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>

```

Run with `xsltproc positions.xml egf-sample.xml > positions.csv`. Sample output:

```

ISIN,Name,Currency,MarketValue
DE0007164600,SAP SE,EUR,7450000.00
NL0010273215,ASML Holding N.V.,EUR,9972200.00
CH0038863350,Nestle S.A.,CHF,6017500.00

```

The `xsl:key` on line 5 indexes all position-type children of `<Position>` by their `<UniqueID>` — the same `xs:IDREF` mechanism that Chapter 6 described. The main template iterates over the `Asset` entries in `AssetMasterData`, looks up the matching position via `key('pos', UniqueID)`, and emits the combined fields as comma-separated text. This join pattern — master data on one side, positions on the other, linked by `UniqueID` — is the single most important XSLT pattern for FundsXML work, because the schema's two-container model (assets separate from positions) makes it necessary in almost every extraction task.

Both examples use XSLT 1.0, which is supported by `xsltproc` (bundled with libxml2 on most Linux and macOS systems) and by every Java, Python, and .NET XML stack. For tasks that require grouping, date formatting, regular expressions, or multiple output documents, XSLT 2.0 or 3.0 via Saxon-HE (the free open-source edition) is the recommended upgrade. Chapter 11 §11.3.4 covers the FreeXmlToolkit XSLT Developer tab, which uses Saxon as its engine and provides a live-preview environment for developing and debugging XSLT stylesheets against FundsXML files.

More extensive XSLT stylesheets, together with Schematron rules, sample XML files, and converter configurations, are maintained in the community FundsXML examples repository at <https://github.com/fundxml/examples>. Appendix E §E.1 has the full reference entry.

12.4 Reading and Processing Strategies

XML libraries in every language offer three fundamentally different ways to read a document: **DOM** (build the whole tree in memory), **streaming** (process events as the parser emits them), and **XPath** (query a built tree with a path expression). Each strategy has strengths and weaknesses, and the right one depends on the size of the input, the access pattern, and the available memory.

12.4.1 DOM: The Whole Tree in Memory

DOM is the simplest approach: the parser reads the entire document and builds a tree of `Element` / `Attribute` / `Text` nodes in memory, and the application navigates the tree with standard methods. Every example in §12.3 used DOM parsing.

When DOM is the right choice:

- The file is small to medium — up to a few megabytes, comfortably fits in memory without strain.
- The access pattern is random — the application needs to jump between distant parts of the document (read `ControlData`, then jump to a regulatory reporting block, then back to a fund definition).
- Development speed matters more than runtime efficiency.

When DOM is the wrong choice:

- The file is large — tens of megabytes of FundsXML data, with thousands of positions and many `RegulatoryReportings` entries. DOM parsers typically consume 3–10 times the file size in RAM, so a 50 MB FundsXML file can easily need 300 MB of memory to hold.

- Memory is constrained — a consumer pipeline running in a resource-limited container may not have the room for a DOM build.
- The application needs to start emitting results before the whole document is parsed (streaming output from streaming input).

For the vast majority of FundsXML files — month-end deliveries for a single fund, regulatory disclosures for a handful of share classes — DOM is the right choice and the strategies below are unnecessary overhead. The alternatives become interesting at scale.

12.4.2 Streaming: SAX and StAX

Streaming parsers read the document once, from start to end, and emit events as they encounter elements, attributes, and text. The application processes each event as it arrives and then discards it; only the information the application chooses to keep lives in memory. A 500 MB FundsXML file can be processed in constant memory if the application is careful.

Two streaming APIs dominate:

- **SAX (Simple API for XML)** is *push-based*: the parser drives the processing by calling handler methods on a user-supplied callback object (`startElement` , `endElement` , `characters`). The application structure is inverted from the usual call-a-function model — the parser calls into the application, not the other way around.
- **StAX (Streaming API for XML)** is *pull-based*: the application calls `next()` on a parser object to retrieve the next event and handles it inline. The application structure is conventional — the app drives the parser — at the cost of the application holding a slightly larger state machine.

Both models have the same *performance characteristics* (constant memory, $O(n)$ time in the file size) but differ in *programming style*. SAX is harder to write correctly for anything beyond trivial transformations, because the callback discipline forces the application to track its own state as it walks the event stream. StAX is easier for non-trivial transformations because the pull model lets the developer write normal sequential code.

The typical FundsXML use case for streaming is a consumer loader that reads a large administrator batch file containing many funds. The loader iterates over the `<Fund>` elements one at a time, extracts the fund's identifiers and dynamic data, inserts them into a database, and then releases the fund's memory before moving to the next one. A DOM-based version of the same loader would hold the entire batch file in memory throughout the load.

12.4.3 XPath: Query-Based Access

XPath is a query language for XML trees. Rather than navigating through explicit method calls (`.Element("Funds").Element("Fund").Element("Identifiers").Element("LEI")`), the developer writes a path expression (`/FundsXML4/Funds/Fund/Identifiers/LEI`) and the XPath engine evaluates it against the document. The result is either a single node, a list of nodes, a boolean, a number, or a string, depending on the expression.

XPath is built on top of DOM: the document must already be parsed into a tree before XPath can query it. So XPath does not help with the large-file memory problem, but it helps with two other problems:

- **Concise navigation.** Deeply nested paths that would be ugly chains of `.Element()` calls become single path expressions.
- **Ad-hoc exploration.** Interactive XPath evaluation (in FreeXmlToolkit, in an IDE debugger, or in the Python REPL) is the fastest way to answer "is this field populated in this file?" without writing navigation code.

The Java example in §12.3.3 used XPath for the deepest path (the one down to the `Amount` element). The Python example used `lxml's find()`, which takes a subset-XPath syntax. Both are examples of XPath replacing longer navigation chains.

XPath has two version families — XPath 1.0, which is universally supported, and XPath 2.0/3.0, which is significantly more powerful but requires a more capable processor (Saxon, for example). For most FundsXML tasks, XPath 1.0 is enough; for tasks that need sequence operations, typed comparisons, or regular expressions, XPath 2.0 is worth the extra dependency.

12.4.4 When to Use Which

Table 12.2 — Reading strategies for different FundsXML workloads

Workload	Strategy	Notes
Single fund delivery, month-end	DOM	Simplest, file fits comfortably in memory
Interactive exploration / debugging	DOM + XPath	XPath queries feel like SQL
Administrator batch, many funds	StAX	Stream and release memory per fund
Very large archive, historical load	SAX or StAX	Constant memory
Subset extraction (one field from many files)	Streaming with early termination	Parse until the field is found, then stop
Transformation to another format	XSLT	Native match-and-transform semantics

A pragmatic consumer pipeline for a mid-sized distributor typically uses **DOM + XPath** for everything it consumes, because its incoming files are small enough not to stress memory and the development speed matters more than the performance difference. A warehouse loader handling administrator batches typically uses **StAX** or **XSLT** because the files are larger and the load runs frequently enough for the memory and CPU savings to matter.

12.5 Database Integration

FundsXML is a transport format, not a storage format. Every production pipeline that consumes FundsXML eventually writes the data somewhere — into a database, a data lake, a warehouse, or occasionally a filesystem archive — and the choice of target storage is one of the most consequential architectural decisions in the pipeline.

Three broad strategies exist, and they correspond to three different database paradigms. To make them concrete, all three examples in this section work from the same FundsXML fragment — a minimal extract of the Europa Growth Fund's month-end delivery with three portfolio positions:

```
<Funds>
  <Fund>
    <Identifiers><LEI>549300ABCDEFGHIIJ1234</LEI></Identifiers>
    <Names><OfficialName>Europa Growth Fund</OfficialName></Names>
    <Currency>EUR</Currency>
    <SingleFundFlag>true</SingleFundFlag>
    <FundDynamicData>
      <TotalAssetValues>
        <TotalAssetValue>
          <NavDate>2026-03-31</NavDate>
          <TotalAssetNature>OFFICIAL</TotalAssetNature>
          <TotalNetAssetValue>
            <Amount ccy="EUR">464552848.78</Amount>
          </TotalNetAssetValue>
        </TotalAssetValue>
      </TotalAssetValues>
      <PortfolioData>
        <Portfolio>
          <Position>
            <Identifiers><ISIN>DE0007236101</ISIN></Identifiers>
            <Name>Siemens AG</Name>
            <Quantity>42500</Quantity>
            <MarketValue ccy="EUR">7203750.00</MarketValue>
          </Position>
          <Position>
            <Identifiers><ISIN>FR0000121014</ISIN></Identifiers>
            <Name>LVMH Moët Hennessy</Name>
            <Quantity>8200</Quantity>
            <MarketValue ccy="EUR">5576000.00</MarketValue>
          </Position>
          <Position>
            <Identifiers><ISIN>CH0038863350</ISIN></Identifiers>
            <Name>Nestlé S.A.</Name>
            <Quantity>61000</Quantity>
            <MarketValue ccy="EUR">5795000.00</MarketValue>
          </Position>
        </Portfolio>
      </PortfolioData>
    </FundDynamicData>
```

```
</Fund>
</Funds>
```

12.5.1 Relational Shredding

The classic approach is to **shred** the FundsXML document into rows and columns in a relational database. A `Fund` element becomes a row in the `funds` table; each `ShareClass` becomes a row in the `share_classes` table linked to the fund by foreign key; each portfolio position becomes a row in the `positions` table linked to the fund and to an asset master record; each `SingleFundFlow` (or its real-schema equivalent) becomes a row in the `transactions` table. The shredding is one-directional: once the file has been decomposed, it lives as rows, and the original XML is either archived or discarded.

Strengths:

- SQL is the universal query language of the data world, and shredding makes FundsXML data queryable by every tool that speaks SQL: BI platforms, dashboards, Excel's Power Query, Python pandas with SQLAlchemy, Java JDBC.
- Relational aggregation (`SUM` , `GROUP BY` , joins) is fast and predictable once the data is normalised.
- Mature constraint enforcement: foreign keys, unique constraints, and `CHECK` constraints can enforce data integrity at load time.
- The operational ecosystem around relational databases (backups, replication, monitoring) is the most mature of any data store.

Weaknesses:

- The shredding step is expensive to design and maintain. Every new field in the FundsXML schema needs a new column or a new table; every schema upgrade needs a corresponding migration.
- Optional and sparse fields (which FundsXML has many of) become `NULL` columns, which bloat tables and can confuse naive consumers.
- Relationships that are natural in XML (multi-language text, hierarchical decomposition) become awkward in SQL (one row per language, recursive joins).
- Re-constructing the original FundsXML document from the shredded rows is non-trivial and often lossy.

When to choose: the consumer is a data warehouse, a regulatory analytics platform, or a BI tool that needs SQL access. The source data is relatively stable and the schema evolution is managed.

Example — PostgreSQL with Python

The schema below captures the fund master data and the portfolio positions in two normalised tables. In a corporate environment the DDL is typically managed separately — by a DBA or a migration tool such as Alembic or Flyway — so we show it as standalone SQL.

```
CREATE TABLE funds (
    fund_id      SERIAL PRIMARY KEY,
    lei          VARCHAR(20)  NOT NULL UNIQUE,
    name         VARCHAR(256) NOT NULL,
```

```

        currency    CHAR(3)          NOT NULL,
        nav_date    DATE,
        nav_amount  NUMERIC(18,2)
    );

    CREATE TABLE positions (
        position_id  SERIAL PRIMARY KEY,
        fund_id      INTEGER NOT NULL REFERENCES funds(fund_id),
        nav_date     DATE     NOT NULL,
        isin         CHAR(12) NOT NULL,
        name         VARCHAR(256),
        quantity     NUMERIC(18,4),
        market_value NUMERIC(18,2),
        currency     CHAR(3)
    );

```

The import script reads the FundsXML file with `lxml`, extracts the fund and position data by XPath, and inserts it into PostgreSQL with parametrised queries. Install the dependencies with `pip install lxml psycopg`.

```

# import_fund.py – FundsXML to PostgreSQL
from lxml import etree
import psycopg
import sys

def import_fundxml(path, conn_string):
    tree = etree.parse(path)
    fund = tree.find("./Fund")
    lei    = fund.findtext("Identifiers/LEI")
    name   = fund.findtext("Names/OfficialName")
    currency = fund.findtext("Currency")
    nav    = fund.find("./TotalAssetValue")
    nav_date = nav.findtext("NavDate")
    nav_amt = nav.findtext("TotalNetAssetValue/Amount")

    with psycopg.connect(conn_string) as conn:
        with conn.cursor() as cur:
            cur.execute("""
                INSERT INTO funds (lei, name, currency, nav_date, nav_amount)
                VALUES (%s, %s, %s, %s, %s)
                RETURNING fund_id
            """, (lei, name, currency, nav_date, nav_amt))
            fund_id = cur.fetchone()[0]

            for pos in fund.findall("./Portfolio/Position"):
                cur.execute("""
                    INSERT INTO positions
                    (fund_id, nav_date, isin, name,
                     quantity, market_value, currency)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, (fund_id, nav_date,

```



```

        pos.findtext("Identifiers/ISIN"),
        pos.findtext("Name"),
        pos.findtext("Quantity"),
        pos.findtext("MarketValue"),
        currency))
    conn.commit()
    print(f"Imported {lei}: fund_id={fund_id}")

if __name__ == "__main__":
    import_fundxml(sys.argv[1], "postgresql://localhost/fundxml")

```

With the data in the database, the reverse direction reconstructs a FundsXML fragment from the relational rows. The export script queries the two tables and builds the XML tree with `lxml.etree`.

```

# export_fund.py – PostgreSQL to FundsXML
from lxml import etree
import psycopg2
import sys

def export_fundxml(lei, conn_string, output_path):
    with psycopg2.connect(conn_string) as conn:
        with conn.cursor() as cur:
            cur.execute(
                "SELECT name, currency, nav_date, nav_amount "
                "FROM funds WHERE lei = %s", (lei,))
            f = cur.fetchone()
            cur.execute(
                "SELECT isin, name, quantity, market_value "
                "FROM positions p JOIN funds f ON f.fund_id = p.fund_id "
                "WHERE f.lei = %s ORDER BY p.market_value DESC", (lei,))
            positions = cur.fetchall()

            fund = etree.Element("Fund")
            ids = etree.SubElement(fund, "Identifiers")
            etree.SubElement(ids, "LEI").text = lei
            names = etree.SubElement(fund, "Names")
            etree.SubElement(names, "OfficialName").text = f[0]
            etree.SubElement(fund, "Currency").text = f[1]
            dyn = etree.SubElement(fund, "FundDynamicData")
            tavs = etree.SubElement(dyn, "TotalAssetValues")
            tav = etree.SubElement(tavs, "TotalAssetValue")
            etree.SubElement(tav, "NavDate").text = str(f[2])
            etree.SubElement(tav, "TotalAssetNature").text = "OFFICIAL"
            tnav = etree.SubElement(tav, "TotalNetAssetValue")
            amt = etree.SubElement(tnav, "Amount", ccy=f[1])
            amt.text = str(f[3])
            port = etree.SubElement(
                etree.SubElement(dyn, "PortfolioData"), "Portfolio")
            for isin, name, qty, mv in positions:
                pos = etree.SubElement(port, "Position")

```

```

    p_ids = etree.SubElement(pos, "Identifiers")
    etree.SubElement(p_ids, "ISIN").text = isin
    etree.SubElement(pos, "Name").text = name
    etree.SubElement(pos, "Quantity").text = str(qty)
    etree.SubElement(pos, "MarketValue", ccy=f[1]).text = str(mv)

    tree = etree.ElementTree(fund)
    tree.write(output_path, pretty_print=True,
              xml_declaration=True, encoding="UTF-8")
    print(f"Exported {lei} to {output_path}")

if __name__ == "__main__":
    export_fundxml(sys.argv[1], "postgresql://localhost/fundxml",
                  sys.argv[2])

```

A production version would wrap this in a full `<FundsXML4>` envelope with `<ControlData>`, add connection pooling and error recovery, and handle the many optional fields that the simplified example omits. The pattern, however, is the same: query the relational tables, build the XML tree, serialise.

Once the data is loaded, the full power of SQL is available. A typical analytical query — the top positions by market value as a percentage of NAV:

```

SELECT p.isin, p.name, p.market_value,
       ROUND(p.market_value / f.nav_amount * 100, 2) AS pct_of_nav
FROM positions p
JOIN funds f ON f.fund_id = p.fund_id
WHERE f.lei = '549300ABCDEFGHIJ1234'
ORDER BY p.market_value DESC;

```

isin	name	market_value	pct_of_nav
DE0007236101	Siemens AG	7203750.00	1.55
CH0038863350	Nestlé S.A.	5795000.00	1.25
FR0000121014	LVMH Moët Hennessy	5576000.00	1.20

Extending this schema to accommodate share classes, transactions, or regulatory modules follows the same pattern: one table per FundsXML element type, linked by foreign keys. The trade-off is clear — every new FundsXML field requires a migration, but once in SQL, the data is accessible from any tool in the organisation.

12.5.2 JSON / Document Database

A modern alternative is to **convert** the FundsXML document to JSON and store it as a document in a document database (MongoDB, Couchbase, DynamoDB, PostgreSQL's JSONB). The document keeps its hierarchical structure; queries navigate through JSON path expressions rather than SQL joins; each FundsXML delivery becomes one or a handful of JSON documents in a collection.

Strengths:

- Schema evolution is free: a new field in FundsXML becomes a new property in the JSON document, and the document database accommodates it without migration.
- The hierarchy is preserved: a fund with its share classes, its portfolio, and its regulatory reportings all travel together as one logical unit.
- Good fit for applications that read whole documents at a time (a distributor's portal showing one fund's profile, a risk system loading the latest regulatory submission).
- JSON is a more natural serialisation format for modern web and mobile applications than XML.

Weaknesses:

- Cross-document queries (give me all funds with NAV > 100 million) are possible but slower than in a relational database.
- Aggregation (sum the total net assets across all European funds) works but is less efficient than relational SQL with indexed columns.
- XML-to-JSON conversion is lossy in subtle ways: attributes vs elements, mixed content, ordered-vs-unordered children. A careless conversion can drop information that the consumer later needs.
- The tooling ecosystem, while growing, is less mature than relational tooling.

When to choose: the consumer is an application that reads documents in whole-document units, schema evolution is expected to be frequent, and the data volume is moderate.

Example — MongoDB with Java

The Java import script uses the same JAXP and XPath approach from §12.3.3 to parse the FundsXML file, then builds a nested BSON document for MongoDB. The fund and its positions travel together as a single document — the hierarchical structure of the XML maps naturally to MongoDB's document model. Add `mongodb-driver-sync` (Maven: `org.mongodb:mongodb-driver-sync:5.x`) to the project dependencies.

```
// FundsXmlToMongo.java – FundsXML to MongoDB
import com.mongodb.client.*;
import org.bson.Document;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import java.util.*;

public class FundsXmlToMongo {
    public static void main(String[] args) throws Exception {
        var dbf = DocumentBuilderFactory.newInstance();
        var doc = dbf.newDocumentBuilder().parse(args[0]);
        var xp = XPathFactory.newInstance().newXPath();

        String lei = xp.evaluate("//Fund/Identifiers/LEI", doc);
        String name = xp.evaluate("//Fund/Names/OfficialName", doc);
        String ccy = xp.evaluate("//Fund/Currency", doc);
        String date = xp.evaluate("//TotalAssetValue/NavDate", doc);
        String nav = xp.evaluate("//TotalNetAssetValue/Amount", doc);
```

```

var positions = new ArrayList<Document>();
var nodes = (NodeList) xp.evaluate(
    "-//Portfolio/Position", doc, XPathConstants.NODESET);
for (int i = 0; i < nodes.getLength(); i++) {
    var pos = nodes.item(i);
    positions.add(new Document()
        .append("isin", xp.evaluate("Identifiers/ISIN", pos))
        .append("name", xp.evaluate("Name", pos))
        .append("quantity", Double.parseDouble(
            xp.evaluate("Quantity", pos)))
        .append("marketValue", Double.parseDouble(
            xp.evaluate("MarketValue", pos))));
}

var fund = new Document()
    .append("lei", lei).append("name", name)
    .append("currency", ccy).append("navDate", date)
    .append("navAmount", Double.parseDouble(nav))
    .append("positions", positions);

try (var client = MongoClient.create(
    "mongodb://localhost:27017")) {
    client.getDatabase("fundxml")
        .getCollection("funds")
        .insertOne(fund);
    System.out.println("Imported " + lei);
}
}
}

```

The export script reverses the process: it reads the document from MongoDB and reconstructs a FundsXML fragment using the JAXP DOM builder and the `Transformer` serialiser.

```

// MongoToFundsXml.java – MongoDB to FundsXML
import com.mongodb.client.*;
import static com.mongodb.client.model.Filters.eq;
import org.bson.Document;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.*;
import java.util.List;

public class MongoToFundsXml {
    public static void main(String[] args) throws Exception {
        Document fund;
        try (var client = MongoClient.create(
            "mongodb://localhost:27017")) {
            fund = client.getDatabase("fundxml")

```

```

        .getCollection("funds")
        .find(eq("lei", args[0])).first();
    }

    var dbf = DocumentBuilderFactory.newInstance();
    var doc = dbf.newDocumentBuilder().newDocument();
    var root = doc.createElement("Fund");
    doc.appendChild(root);
    appendTextElement(doc, appendElement(doc, root,
        "Identifiers"), "LEI", fund.getString("lei"));
    appendTextElement(doc, appendElement(doc, root,
        "Names"), "OfficialName", fund.getString("name"));
    appendTextElement(doc, root, "Currency",
        fund.getString("currency"));
    var dyn = appendElement(doc, root, "FundDynamicData");
    var tav = appendElement(doc, appendElement(doc, dyn,
        "TotalAssetValues"), "TotalAssetValue");
    appendTextElement(doc, tav, "NavDate",
        fund.getString("navDate"));
    var amt = appendTextElement(doc, appendElement(doc, tav,
        "TotalNetAssetValue"), "Amount",
        String.valueOf(fund.getDouble("navAmount")));
    amt.setAttribute("ccy", fund.getString("currency"));
    var port = appendElement(doc, appendElement(doc, dyn,
        "PortfolioData"), "Portfolio");
    for (var p : fund.getList("positions", Document.class)) {
        var pos = appendElement(doc, port, "Position");
        appendTextElement(doc, appendElement(doc, pos,
            "Identifiers"), "ISIN", p.getString("isin"));
        appendTextElement(doc, pos, "Name",
            p.getString("name"));
        appendTextElement(doc, pos, "Quantity",
            String.valueOf(p.getDouble("quantity").intValue()));
        var mv = appendTextElement(doc, pos, "MarketValue",
            String.valueOf(p.getDouble("marketValue")));
        mv.setAttribute("ccy", fund.getString("currency"));
    }

    var tf = TransformerFactory.newInstance().newTransformer();
    tf.setOutputProperty(OutputKeys.INDENT, "yes");
    tf.transform(new DOMSource(doc),
        new StreamResult(new File(args[1])));
    System.out.println("Exported " + args[0] + " to " + args[1]);
}

static org.w3c.dom.Element appendElement(
    org.w3c.dom.Document doc,
    org.w3c.dom.Element parent, String name) {
    var el = doc.createElement(name);
    parent.appendChild(el);
    return el;
}

```

```

static org.w3c.dom.Element appendTextElement(
    org.w3c.dom.Document doc,
    org.w3c.dom.Element parent, String name, String text) {
    var el = appendElement(doc, parent, name);
    el.setTextContent(text);
    return el;
}
}

```

The verbosity of the Java DOM builder compared to Python's `lxml.etree.SubElement()` is characteristic: Java requires more ceremony, but the code is type-safe and runs on every JVM without external dependencies beyond the MongoDB driver. A production version would add error handling, use a `MongoCredential` for authentication, and wrap the export in a full `<FundsXML4>` envelope with `<ControlData>`.

12.5.3 XML-Native Databases

A third option is to store the FundsXML documents in an **XML-native database** — a database designed to hold XML documents without conversion, query them with XPath or XQuery, and preserve the full XML semantics (namespaces, validation, identity constraints). The open-source leaders in this category are **BaseX** and **eXist-db**; commercial options include **MarkLogic** and **Oracle XML DB**.

Strengths:

- No conversion loss: the document is stored as XML, and consumers retrieve it as XML. Round-tripping is perfect.
- Queries use XPath 2.0 / XQuery, which are purpose-built for XML and more powerful than XPath 1.0. Complex transformations are concise.
- Schema validation can happen at load time, so the database holds only valid documents.
- Useful for legal and regulatory archiving, where the original document form matters for compliance.

Weaknesses:

- Smaller operational ecosystem: fewer DBAs, fewer tools, fewer commercial vendors, less documentation than for relational or document databases.
- Query performance can be unpredictable for complex XPath expressions, and optimisation requires XQuery expertise that few teams have.
- Integration with non-XML tools (dashboards, reporting engines, machine-learning pipelines) is awkward because most consuming tools expect SQL or JSON.
- The learning curve for XQuery is steep for developers coming from SQL backgrounds.

When to choose: the consumer is a regulatory archive, a legal-evidence store, or a specialised XML processing application where the round-trip fidelity matters more than integration with non-XML tools.

Example — BaseX with Java

BaseX is a lightweight, open-source XML database written in Java. Because it is Java-native, the most natural integration path is the embedded Java API (`org.basex:basex` from Maven Central), which gives the application direct access to the database engine without a separate server process.

The import script is deliberately short — that is the point. An XML-native database stores the document as-is; there is no shredding, no conversion, no mapping.

```
// FundsXmlToBaseX.java – FundsXML to BaseX
import org.basex.core.*;
import org.basex.core.cmd.*;

public class FundsXmlToBaseX {
    public static void main(String[] args) throws Exception {
        try (var ctx = new Context()) {
            new CreateDB("fundxml", args[0]).execute(ctx);
            System.out.println("Imported " + args[0]
                + " into database 'fundxml'");
        }
    }
}
```

Four lines of business logic — compare that with the forty lines of the PostgreSQL import or the fifty lines of the MongoDB import. The brevity is the argument: where the data is already XML, an XML-native database eliminates the impedance mismatch.

The query and export script opens the database and runs XQuery expressions against it. The first query extracts the fund and its positions as a new XML document; the second computes portfolio weights. Both queries navigate the original FundsXML structure directly — no ORM, no mapping layer, no intermediate representation.

```
// BaseXQuery.java – Query and export from BaseX
import org.basex.core.*;
import org.basex.core.cmd.*;

public class BaseXQuery {
    public static void main(String[] args) throws Exception {
        try (var ctx = new Context()) {
            new Open("fundxml").execute(ctx);

            // Export: extract fund and positions as XML
            String export = new XQuery("""
                for $fund in //Fund
                let $name := $fund/Names/OfficialName/text()
                let $lei := $fund/Identifiers/LEI/text()
                let $nav := $fund/FundDynamicData/TotalAssetValues
                        /TotalAssetValue
                        /TotalNetAssetValue/Amount/text()
                return
                    <Fund>
                        <Identifiers><LEI>{$lei}</LEI></Identifiers>
            """).execute(ctx);
        }
    }
}
```

```

        <Names>
          <OfficialName>{$name}</OfficialName>
        </Names>
      {
        for $pos in $fund/FundDynamicData
          /PortfolioData/Portfolio/Position
        return $pos
      }
    </Fund>
    """).execute(ctx);
    System.out.println(export);

    // Analytical query: portfolio weights
    String weights = new XQuery("""
      let $fund := //Fund
      let $nav := xs:decimal($fund/FundDynamicData
        /TotalAssetValues/TotalAssetValue
        /TotalNetAssetValue/Amount)
      for $pos in $fund/FundDynamicData
        /PortfolioData/Portfolio/Position
      let $mv := xs:decimal($pos/MarketValue)
      order by $mv descending
      return concat($pos/Identifiers/ISIN, ' ',
        $pos/Name, ' ',
        round($mv div $nav * 10000)
        div 100, '%')
    """).execute(ctx);
    System.out.println(weights);
  }
}

```

The export query produces a valid `<Fund>` element that could be wrapped in a `<FundsXML4>` envelope and written to a file — the round-trip is lossless because the data was never converted out of XML. The portfolio-weight query produces:

```

DE0007236101 Siemens AG 1.55%
CH0038863350 Nestlé S.A. 1.25%
FR0000121014 LVMH Moët Hennessy 1.2%

```

The contrast with the relational and document-database examples is instructive: no schema migration, no type conversion, no mapping code. The trade-off is that the analytical query runs inside XQuery rather than SQL, which limits the tooling ecosystem that can consume the results. Adding new queries for share classes or regulatory modules requires only new XQuery expressions, never a schema migration.

12.5.4 Hybrid Architectures

In practice, many mature FundsXML deployments use **two or three** storage strategies in combination. A typical arrangement:

- **Relational warehouse** for analytics and reporting, populated by a shredding ETL job.
- **Document database** for the operational application layer (distributor portals, fund-profile pages), populated by the same ETL.
- **XML-native archive** (or simple filesystem archive with the original XML files plus an index) for audit, regulatory evidence, and the rare case where reconstructing the original document matters.

The three targets are loaded from the same incoming FundsXML files, typically through parallel pipelines that fan out from a common ingestion layer. Each target optimises for its own use case; the overall system benefits from the strengths of each without being limited by any one of them.

12.6 Data Warehousing and ETL Patterns

A specific case of the "relational shredding" strategy from §12.5.1 deserves its own section: **loading FundsXML into a data warehouse** for historical and analytical purposes. Warehouses are the long-term memory of the fund industry — they hold years of portfolio snapshots, NAV histories, transaction flows, and regulatory disclosures, organised so that analysts can query across time and across funds.

12.6.1 Star Schemas and Fact Tables

The dominant warehouse schema for FundsXML data is the **star schema**: a central **fact table** holding the per-fund or per-position numeric measures, surrounded by **dimension tables** holding the categorical descriptors. A typical fund-level warehouse might have:

- **Fund dimension** — one row per fund, with identifiers, names, domicile, inception date, and other static attributes.
- **Share class dimension** — one row per share class, with ISIN, currency, distribution policy, and so on.
- **Date dimension** — one row per day, with fiscal-year and calendar attributes (the classic warehouse date-dimension trick).
- **Asset dimension** — one row per instrument, with ISIN, name, sector, country, and so on.
- **Fact: daily NAV** — one row per (share class, day), with NAV, shares outstanding, total net assets.
- **Fact: monthly positions** — one row per (fund, date, asset), with quantity and market value.
- **Fact: daily flows** — one row per (fund, date, flow type), with amounts.

A FundsXML consumer pipeline shreds each incoming delivery into these fact-table rows, using the dimension tables as lookup sources. Analysts then query the warehouse with standard SQL joins to answer questions like "what was the total net assets of all German equity funds at the end of each quarter last year?".

12.6.2 ETL Pipeline Patterns

The ETL (Extract, Transform, Load) pipeline that feeds the warehouse typically follows one of three patterns.

Pattern 1 — Batch ETL. The pipeline runs on a fixed schedule (nightly, weekly), reads every FundsXML file received since the last run, shreds each file into fact and dimension rows, and bulk-inserts them into the warehouse. Batch ETL is simple, easy to audit, and easy to re-run when something goes wrong. Its weakness is latency: data in a file received at 14:00 is not queryable in the warehouse until the batch runs at 23:00. For month-end fund data where the queries are about "yesterday's NAV", this is usually acceptable.

Pattern 2 — Streaming ETL. The pipeline runs continuously, watching for new FundsXML files as they arrive and loading them into the warehouse within minutes of arrival. Streaming ETL is better for applications that need low-latency analytics (a trading desk that wants to see the latest portfolio composition, a risk system that recalculates overnight). It is harder to operate because the pipeline must handle failure recovery in real time, and deduplication becomes more important.

Pattern 3 — Hybrid. A streaming pipeline handles the time-critical subset (ControlData, fund identifiers, today's NAV), and a nightly batch pipeline handles the heavier material (portfolios, regulatory reportings, historical backfills). Many production warehouses use this split because it matches the different latency requirements of different query audiences.

12.6.3 Change Data Capture and Amendments

A specific complication for warehouse loaders: how to handle FundsXML `AMEND` and `DELETE` operations. A warehouse that naively appends every received delivery as new rows will accumulate contradictions over time (the NAV for 31 March 2026 as of the first delivery, the NAV for 31 March 2026 as of the corrected delivery, and so on). The warehouse needs to know which version is currently authoritative.

Two approaches solve this. The **snapshot approach** maintains a "current state" table that is overwritten for each new delivery, alongside a history table that appends every version ever seen. Queries against "current" data use the first table; queries against "as-of-a-given-date" data use the second. The **effective-dating approach** tags every row with an `effective_from` / `effective_to` date pair, so that queries can specify a point in time and the warehouse returns the version that was authoritative at that time. Both approaches have trade-offs; Chapter 13 will revisit this in the context of full implementation-project design.

12.7 Automation and Scheduling

A FundsXML pipeline is rarely run by hand. Once the pipeline is working, the question shifts from "how does it work?" to "how does it run reliably without human intervention?" — and that is the domain of automation and scheduling.

12.7.1 Simple Scheduling — cron and Windows Task Scheduler

The simplest automation is a time-based scheduler. On Linux and macOS, `cron` runs a shell command at a specified time; on Windows, the Task Scheduler does the same. For a producer pipeline that emits a monthly delivery at a fixed hour, a one-line cron entry is enough:

```
0 22 28-31 * * /opt/fundxml/bin/emit-monthly.sh
```

(Run at 22:00 on the 28th through 31st of every month, with a logic inside the script to determine whether the current date is the last business day.)

Simple scheduling works well for pipelines with:

- **Fixed cadence:** the job runs at the same time every day/week/month, and a missed run is either acceptable or easy to rerun manually.
- **Short duration:** the job finishes within its scheduled window and does not risk overlapping with the next run.
- **Low dependency complexity:** the job does not need to wait for other jobs to complete before starting.

12.7.2 Event-Driven Triggers

Many consumer pipelines cannot wait for a schedule: they need to react to incoming files as soon as the files arrive. The typical approach is an **event-driven trigger** — a filesystem watcher, a message-queue subscriber, or a webhook endpoint that starts the processing pipeline the moment a new file lands.

On Linux, `inotifywait` watches a directory and triggers a script for every new file. On cloud platforms, the equivalent is an object-storage event (S3 bucket notification, Azure Blob Storage trigger, Google Cloud Storage notification) that invokes a serverless function. On message-broker-based architectures, a subscriber on a Kafka topic or a RabbitMQ queue consumes delivery events as they are published.

The operational trade-off compared to scheduling is lower latency (the pipeline starts immediately) at the cost of higher complexity (the pipeline must handle concurrent triggers, retry on failure, and guarantee idempotent processing). For a distributor dispatcher that needs to route incoming files to the right internal systems within minutes, event-driven triggers are essential.

12.7.3 Workflow Orchestrators

Pipelines with complex dependencies between steps — "first validate the file, then shred it, then load dimension rows, then load facts, then run the business-rule checks, then notify downstream" — outgrow simple scheduling and benefit from a **workflow orchestrator**. The major options:

- **Apache Airflow** — the open-source standard for Python-based data pipelines. Workflows are defined as Python code (DAGs of tasks), and the Airflow web UI shows status, history, and logs. Widely used in data-engineering teams.
- **Prefect** and **Dagster** — modern alternatives to Airflow, with cleaner APIs and better local-development experience.

- **Apache NiFi** — a flow-based tool designed for data routing and transformation, with a visual flow-designer UI. Common in enterprise IT.
- **Azure Data Factory, AWS Step Functions, Google Cloud Workflows** — cloud-native managed equivalents.

A typical Airflow DAG for a FundsXML producer pipeline might have tasks for: aggregate source data, generate FundsXML, validate XSD, validate Schematron, sign (if required), emit to distributor drop-box, log to audit trail, notify on success, notify on failure. Each task is a Python function; dependencies between tasks are declared explicitly; retries on failure are configured per task; the whole DAG runs on the schedule defined in Airflow.

Orchestrators shine when pipelines have more than a handful of steps, when failure recovery matters, and when multiple teams need visibility into pipeline status. For simple single-step pipelines they are overkill.

12.7.4 Choosing the Right Approach

Table 12.3 — Automation options for FundsXML pipelines

Requirement	Best fit
Fixed monthly delivery, one script	cron / Task Scheduler
React to incoming files, low latency	Filesystem watcher / cloud event
Multi-step pipeline with dependencies	Airflow / Prefect / Dagster
Cloud-native deployment	AWS Step Functions / Azure Data Factory
Visual pipeline design for non-programmers	Apache NiFi

Most production FundsXML pipelines use a combination: cron for the outermost schedule, event-driven triggers for the consumer side, and an orchestrator for the multi-step processing that happens inside. Chapter 13 will describe a complete implementation project that combines all three.

12.8 Common Pitfalls

- **Choosing a language based on theoretical elegance rather than the existing stack.** The team's existing fluency dominates every minor language advantage. A Java shop that switches to Python for "elegance" spends six months on the migration and another six paying for the resulting mixed-stack maintenance overhead.
- **Using DOM for files that do not fit in memory.** A 500 MB administrator batch file will crash a DOM-based loader on a modestly-provisioned container. Switch to StAX or another streaming approach at the size threshold where DOM starts to struggle, not after the first out-of-memory incident.
- **Ignoring streaming for a job that should use it.** The opposite mistake: using DOM when streaming would have been dramatically faster. If the job reads one small piece of each file (extract the fund's LEI from 10,000 files), streaming with early termination is orders of magnitude faster than full DOM parsing.

- **Shredding FundsXML into a relational schema that mirrors the XML structure too closely.** A relational schema optimised for XML-to-rows mechanical translation is usually poorly suited to analytical queries. Design the warehouse schema for *queries*, not for round-trip fidelity, and accept that re-constructing the original XML from the warehouse is the job of the archive, not the warehouse.
- **Losing information in XML-to-JSON conversion.** A naive converter drops attribute information, collapses mixed content, or loses the ordering of child elements. Use a converter that preserves attributes (as specially-named properties) and document ordering, and test the round-trip against representative files before trusting it.
- **Running Schematron validation only in the producer.** A consumer that trusts the producer's validation entirely has no defence when the producer has a bug. Run the same Schematron rules on both sides.
- **Mixing schedule-based and event-driven triggers without clear boundaries.** If the same pipeline is sometimes triggered by cron and sometimes by an incoming file, locking and idempotency become critical. Pick one trigger model per pipeline and use a lockfile or a database-level flag to prevent concurrent runs.
- **Deploying an orchestrator for a one-step pipeline.** Airflow is overkill for a cron job that runs a single shell script once a day. Save the orchestrator for pipelines that actually have dependencies between steps.

12.9 Key Takeaways

- Production FundsXML deployments fall into a handful of recurring architecture patterns: the **producer pipeline**, the **consumer pipeline**, the **distributor dispatcher**, the **warehouse loader**, and the mixed-workflow asset manager that runs several of the above in parallel.
- All four dominant programming languages — **Java**, **Python**, **C#**, **JavaScript (Node.js)** — can produce and consume FundsXML competently. The right choice is the language the team already uses; the language-specific differences are minor compared to the productivity cost of maintaining code in an unfamiliar language.
- **Python with lxml** is the shortest to write and the fastest to prototype. **Java with JAXP** is the most verbose but the most integrated with enterprise producer systems, and stays inside the JDK's own XML stack without external dependencies. **C# with LINQ to XML** sits in the middle and fits Microsoft-shop asset managers. **Node.js with fast-xml-parser** is the natural fit for distributor-side retail platforms.
- Three reading strategies cover every FundsXML workload: **DOM** for simplicity and random access, **streaming (SAX or StAX)** for large files and constant-memory processing, **XPath** for query-based access on top of DOM. The right choice depends on file size and access pattern.
- Three database paradigms serve FundsXML data differently: **relational shredding** for SQL-queryable warehouses, **document databases (JSON)** for operational application stores, **XML-native databases** for round-trip fidelity archives. Mature deployments often use two or three in combination.
- Warehouse loaders typically follow a **star-schema** design with fact tables for NAVs, positions, and flows, plus dimension tables for funds, share classes, dates, and assets. **Amendments and deletions** require explicit handling, either through snapshot tables or through effective-dating columns.
- Automation ranges from **cron** for simple fixed-schedule pipelines, through **event-driven triggers** for low-latency consumer ingestion, to **workflow orchestrators (Airflow, Prefect, Dagster)** for multi-step pipelines with dependencies. Most production deployments use a combination.

With the system landscape in mind, the last question is practical: *how does an asset manager or distributor actually run an implementation project from start to finish?* Chapter 13 answers that question in detail, walking through the full lifecycle of a FundsXML implementation project for the Europa Growth Fund — from requirements analysis, through mapping, prototyping, testing, and go-live, to ongoing operation and maintenance.



PART III — IMPLEMENTATION AND PRACTICE · CHAPTER 13

An Implementation Project from A to Z

A practical guide for introducing FundsXML

13.1 Setting the Scene: What an Implementation Project Looks Like

Thirteen chapters of schema, tooling, and architecture are useful only insofar as they help a real team ship a real pipeline. This chapter is about that shipping process. It describes, phase by phase, how an asset management company actually introduces FundsXML into its operational landscape — not as a collection of abstract project-management principles, but as a concrete nine-month project at the Europa Asset Management S.A. fictional-but-realistic team that has been the producer of the Europa Growth Fund's data throughout the book.

The project we follow runs from December 2025, when the project kickoff meeting was held, to August 2026, when the first production FundsXML delivery reached all eleven of the fund's distribution countries. In between are six phases: requirements analysis, data mapping, prototyping and piloting, testing, acceptance and go-live, and the transition into operations and maintenance. Each phase gets its own section in this chapter, and §13.8 lays out the full timeline as a single page for the reader who wants to see the arc at a glance.

The project is narratively specific but operationally generic. The names (Europa Asset Management, the Europa Growth Fund, the eleven distribution countries) are the fictional ones that the book has used throughout. The scale and sequence of the phases, the kinds of decisions made at each checkpoint, and the recurring challenges are drawn from real FundsXML implementation projects that the authoring community has seen in practice. A reader running a similar project should find the concrete details useful as a template to adapt rather than as a script to follow literally — every project has its own politics, its own technical debt, and its own internal opposition, and no generic walk-through can capture all of it.

A note on scope. This chapter treats the *producer* side of the project — Europa Asset Management produces FundsXML deliveries, and the project is about building the pipeline that produces them. The *consumer* side (what the eleven distribution countries do with the deliveries) is out of scope for this chapter, except where it intersects with the producer's work. Consumers typically run their own, independent implementation projects that look structurally similar but focus on ingestion rather than emission. Readers on the consumer side can read this chapter with minor mental translation; the phase structure is the same, and most of the decisions are analogous.

By the end of this chapter, you should be able to:

- structure a FundsXML implementation project into its six standard phases and plan the duration of each;
- draw up a requirements document for a producer-side project, naming the data sources, the target consumers, and the regulatory scope;
- build a mapping table that connects internal system fields to FundsXML elements;
- run a prototyping and piloting cycle that catches structural problems before production commitment;

- design a testing plan that covers both schema validation and business-rule validation against realistic fixtures;
 - organise acceptance and go-live gates with the producer's stakeholders and the consumer's downstream systems;
 - transition a working pipeline from "project" to "operations" without losing the institutional knowledge the project accumulated.
-

13.2 Phase 1: Project Preparation and Requirements Analysis

13.2.1 The Trigger and the Kickoff

Every implementation project has a trigger — a specific business event that makes "do it" more urgent than "wait and see". For Europa Asset Management, the trigger was a combination of three pressures in late 2025: the French distributor BNP Distribution Services notified Europa that it would stop accepting the legacy CSV format at the end of Q2 2026 and would require FundsXML going forward; the SFDR Level 2 technical standards mandated quarterly EET deliveries from January 2026, and the existing CSV pipeline had no clean path to carry them; and an internal audit had flagged the CSV pipeline's lack of structured validation as a control weakness after a near-miss incident in October 2025. None of the three pressures alone would have justified the project; all three together made it unavoidable.

The kickoff meeting was held on 8 December 2025. Present were the head of Operations (the project sponsor), the head of IT (delivery accountability), a representative from the Fund Administration team (the data owners), a representative from the Compliance team (regulatory sign-off), and two external consultants engaged for schema expertise. The meeting ran for ninety minutes and produced three outputs: a one-page project charter, a list of twelve open questions that the next four weeks would need to answer, and a commitment to return on 5 January 2026 for a requirements review with a first draft of the answers.

13.2.2 The Requirements Document

Over the next three weeks, the project lead drafted the requirements document that would govern the project for the next nine months. A good FundsXML requirements document has seven sections, each shorter than a typical business requirement document because the FundsXML schema itself carries much of the structural detail that would otherwise need to be written down.

1. Scope. Which funds, which share classes, which distribution countries, which consumers, which regulatory modules. For the Europa project: the Europa Growth Fund and its three share classes, eleven distribution countries, initial consumer list of six retail distributors (the others would be added in a subsequent wave), regulatory modules EMT, EPT, EET, and TPT (but not EFT — the producer does not produce EFT).

2. Frequency and Calendar. How often a delivery is produced and for which valuation dates. Europa's decision: daily NAV deliveries to the internal fund administrator systems, monthly consolidated deliveries to external distributors, quarterly EET refreshes, quarterly TPT updates. The daily and monthly were operationally urgent; the EET and TPT were regulatory deadlines.

3. Data Sources. Which internal systems hold which parts of the data the pipeline needs. Europa's landscape had: a portfolio management system (PMS) holding positions and NAVs, a client reference database (CRD) holding fund and share-class static data, a regulatory reporting data mart (RRDM) holding SFDR classifications and PAI values, and an administrator-provided Excel workbook holding EMT template data. Each of the four sources had its own owner, its own release cadence, and its own quirks.

4. Target Consumers and Delivery Channels. Which external parties receive the deliveries, how they receive them, and what acknowledgements the producer needs. Europa's six initial distributors each had their own SFTP drop-box conventions, their own file-naming rules, and their own acknowledgement mechanisms — two used plain SFTP with a signed delivery receipt, three used SFTP with an out-of-band email confirmation, and one used an HTTPS REST API. The pipeline had to support all three.

5. Regulatory Modules. Which FinDatEx templates the pipeline must populate and to which version. Europa: EMT v4.3, EPT v2.4, EET v1.1.2, TPT v7.0 — all current as of 2026. The requirements document also named the ESMA technical standards each template implemented, so that the team had a clear regulatory anchor.

6. Non-Functional Requirements. Performance, availability, monitoring, audit, retention. Europa: deliveries must be emitted within two hours of the valuation point, availability 99.5% of business days, full audit log of every delivery retained for ten years, Monday-to-Friday business-hours monitoring with on-call escalation only for failures during the emission window.

7. Out of Scope. Explicit statements of what the project will *not* do. Europa's out-of-scope list included: no consumer-side pipeline (consumers handle their own ingestion), no changes to the underlying source systems (the PMS and CRD stay as-is; the pipeline reads from them), no EFT support (this producer does not produce EFT), no historical backfill (the pipeline starts with new deliveries only; pre-2026 data stays in the legacy CSV archive).

13.2.3 The Five Open Questions Every Project Answers Early

Beyond the written requirements, every implementation project has a short list of decisions that need to be made explicitly and early, because a wrong answer taken silently will undermine the rest of the project. Europa's five were:

- 1. Build or buy?** Europa decided to build. The alternative — licensing a vendor FundsXML generator — was considered but rejected on grounds of cost and long-term flexibility. A later section (§13.4) describes the technology stack they chose.
- 2. Java, Python, or C#?** Europa was a Java shop for its core systems but had a small Python team for data engineering. The decision: Java for the generator (integrated into the existing fund-administration platform), Python for the ETL that extracts source data and for the validation pipeline. Node.js was not considered, because no consumer piece was being built.
- 3. In-house or cloud?** Europa ran a private datacentre for its production systems and had no public-cloud deployment in scope. The pipeline would run in the existing on-premises infrastructure, with the option to migrate to cloud later if the organisation's overall strategy moved that way. This decision simplified the security and compliance conversation significantly.
- 4. Single-team or multi-team ownership?** A single cross-functional project team of four engineers plus a part-time BA and a part-time QA was chartered. Ownership would transfer to the existing Fund Operations IT team at go-live. The cross-functional structure was chosen to avoid the handoff overhead that a "development team ships, ops team operates" model would create.

5. **Greenfield or side-by-side?** Europa decided to run the new FundsXML pipeline side-by-side with the legacy CSV pipeline for a transition period, with distributors gradually switching over as each one validated their own consumer. The legacy pipeline would be decommissioned once the last distributor had confirmed successful FundsXML ingestion.

13.2.4 Timeline and Budget

The project plan that emerged from requirements analysis had the following shape: 5 January to 31 January 2026 for requirements completion and stakeholder sign-off; 1 February to 28 February for data mapping; 1 March to 30 April for prototyping and piloting; 1 May to 31 May for testing; 1 June to 30 June for user acceptance and go-live preparation; 1 July for first production delivery; ongoing operations thereafter. The timeline included one intentional slack week in each phase, because every FundsXML project the consultants had seen had encountered unexpected delays somewhere, and the question was only *where* the slack would be consumed. The project ultimately ran two weeks longer than plan and consumed every slack week and then some.

The budget was modest for a project of this scope: four engineers for six months full-time, plus the two consultants part-time, plus tooling and infrastructure costs. The total was comfortably under one million euros — not because FundsXML implementations are cheap, but because the existing source systems provided the hard data, and the project scope was tight enough that no upstream system remediation was required.

13.3 Phase 2: Mapping Existing Data to FundsXML

13.3.1 Why Mapping Is the Hardest Phase

Mapping is the phase where the abstract schema meets the concrete internal data, and it is almost always the phase where FundsXML projects discover that the hard work is not XML — it is understanding the source data. Every field that the FundsXML delivery needs has to come from somewhere, and "somewhere" usually means a column in an internal database whose meaning has been implicit for years and that no one has written down precisely. Phase 2 is the phase where the team writes it all down.

Europa's mapping phase ran from 1 February to 28 February 2026 and consumed roughly two engineer-months of effort. It produced three artefacts: a mapping spreadsheet (the core deliverable), a list of data gaps (fields FundsXML requires that no source system currently holds), and a list of data-quality issues (fields that exist in source but are populated inconsistently or incorrectly).

13.3.2 Structure of a Mapping Table

A mapping table has one row per FundsXML field that the pipeline needs to populate. The columns are:

- **FundsXML element path** — the XPath or dotted path from the root to the target element. For example, `ControlData/DataSupplier/LEI` or `Funds/Fund/Identifiers/LEI`.
- **Source system** — which internal system holds the source data. One of PMS, CRD, RRDM, or EMT-Excel for Europa's landscape.

- **Source column or field** — the specific column name or field identifier in the source system. For example, `CRD.LEGAL_ENTITIES.LEI_CODE`.
- **Transformation rule** — any conversion, lookup, or derivation needed between the source value and the target value. For example, "trim to 20 characters", "convert DD/MM/YYYY to ISO", "lookup country code from COUNTRY_ID via COUNTRY dimension".
- **Default value** — what to write if the source is null or missing. Often blank (meaning "omit the optional element"); sometimes a hard-coded literal.
- **Notes** — any peculiarities, known edge cases, or questions for a later round.

Table 13.1 — Excerpt from Europa Asset Management's mapping table (ControlData)

FundsXML path	Source system	Source column	Transformation	Default
ControlData/UniqueDocumentID	(generated)	—	Concatenate: EGF- + YYYYMMDD + - + sequence number	—
ControlData/DocumentGenerated	(system clock)	—	Current UTC timestamp at emission	—
ControlData/Version	(constant)	—	Hard-coded 4.2.8	4.2.8
ControlData/ContentDate	PMS	VALUATION .VAL_DATE	Convert to ISO format	—
ControlData/DataSupplier/SystemCountry	(constant)	—	Hard-coded LU	LU
ControlData/DataSupplier/Short	(constant)	—	Hard-coded EAM	EAM
ControlData/DataSupplier/Name	CRD	COMPANIES .LEGAL_NAME	Where ROLE = 'MANUFACTURER'	—
ControlData/DataSupplier/Type	(constant)	—	Hard-coded IC	IC
ControlData/DataOperation	(computed)	—	Normal delivery: INITIAL ; retry: AMEND	INITIAL
ControlData/Language	(constant)	—	Hard-coded en	en
Funds/Fund/Identifiers/LEI	CRD	FUNDS.LEI _CODE	Where FUND_ID = :current	—
Funds/Fund/Names/OfficialName	CRD	FUNDS.OFFICIAL_NAME	Where FUND_ID = :current	—
Funds/Fund/Currency	CRD	FUNDS.BASE_CURRENCY	ISO 4217 code	—
Funds/Fund/SingleFundFlag	(computed)	—	true if no sub-funds, else false	true
Funds/Fund/FundDynamicData/TotalAssetValues/TotalAssetValue/NavDate	PMS	NAV_HEAD R.VAL_DATE	—	—

FundsXML path	Source system	Source column	Transformation	Default
Funds/Fund/FundDynamicData/TotalAssetValues/TotalAssetValue/TotalAssetNature	(computed)	—	OFFICIAL for final NAV; ESTIMATED for preliminary	OFFICIAL
Funds/Fund/FundDynamicData/TotalAssetValues/TotalAssetValue/TotalNetAssetValue/Amount	PMS	NAV_HEADE R. TNAV	ccy attribute from NAV_HEADER.CCY	—

The excerpt above shows seventeen rows. The full mapping table for the Europa Growth Fund project had approximately 450 rows covering the complete ControlData, Fund, Portfolio, and [RegulatoryReportings](#) blocks. Each row was reviewed by the data owner of the relevant source system and signed off formally. The sign-off was important because the mapping became the authoritative contract between the producer pipeline and the source systems: any change to a source column's meaning or format would need to be reflected in the mapping before the pipeline could handle it.

13.3.3 Handling Data Gaps

The mapping exercise inevitably surfaces **data gaps**: FundsXML fields that the producer needs to populate but that no existing source system holds. Europa's project found twenty-three such gaps, ranging in severity from "trivial, add a constant" to "major, requires new upstream data feed". Three of the more interesting ones:

- **Funds/Fund/FundStaticData/Custodian** — the custodian (depository bank) of the fund. The CRD held the custodian relationship for recent funds but not for older funds that had been migrated from a legacy system in 2018. For the Europa Growth Fund specifically the data was there; for other funds that would be onboarded later, the CRD would need to be backfilled. The decision: phase-1 launch covered only funds with complete custodian data in CRD; the backfill was logged as a follow-on task.
- **RegulatoryReportings/EET/EET_PA1/*** — the PAI values for the EET. These were held in the RRDM but with a quarterly refresh cadence, which conflicted with the monthly EET that the quarterly-only regulatory schedule didn't strictly demand. The decision: emit the most-recent PAI values in every monthly delivery, with an explicit `EET_PA1_AsOfDate` field indicating when the values were computed. Consumers who need real-time PAI values would get stale data; consumers following the standard quarterly cadence would see no difference.
- **Funds/Fund/FundStaticData/FundTexts** — the multi-language marketing and investment-objective text. These existed in the CRD but only in English; the German, French, and Italian translations were maintained in a separate Marketing-managed spreadsheet that had never been integrated with the core systems. The decision: phase-1 launch included English-only text; the multi-language build-out was scheduled for phase 2 of the project in September 2026.

Data gaps are rarely blockers in themselves, but they force the project to make scoping decisions about whether to launch without the missing data, delay until the gaps are filled, or invent a workaround. Europa's approach — ship without the missing data, fill the gaps in follow-on waves — is the most common pragmatic choice.

13.3.4 Data-Quality Issues

Separately from data gaps, the mapping exercise surfaces **data-quality issues** in fields that *do* exist but hold questionable values. Europa found a dozen of these, two of which were operationally significant:

- **LEI values in the CRD were missing the two-character country prefix check digits.** A valid LEI is 20 characters and the last two are check digits; Europa's CRD held 18-character values because an earlier migration had truncated them. The fix was a one-off database update to recompute the check digits from the authoritative GLEIF register, followed by a CRD constraint to prevent the error from recurring.
- **Currency codes in the PMS used internal three-letter codes that mostly matched ISO 4217 but differed for a few cases** (the internal code for the Norwegian krone was `NOK`, matching ISO, but for the Czech koruna it was `CZK` — close but with historical variants in the data). The fix was a lookup table mapping internal codes to ISO codes, implemented as part of the mapping transformation layer.

Both issues would have caused validation failures once the pipeline went live, and discovering them during mapping was exactly the point of doing the mapping phase thoroughly.

13.4 Phase 3: Prototyping and Piloting

13.4.1 Why a Prototype Comes Before a Production Build

Once the requirements are clear and the mapping is signed off, the temptation is to start building the production pipeline directly. Europa's project did *not* do this. Instead, the team spent two months building a **prototype** — a throw-away implementation of the pipeline that would produce real FundsXML files from real source data, but without any of the production-grade concerns (error handling, monitoring, operational hooks, audit logging). The purpose of the prototype was to discover the problems that only become visible when code meets data.

The prototype ran from 1 March to 30 April 2026 and consumed roughly three engineer-months. It used Python (rather than the Java target for production) because Python was faster for exploratory development; the team knew they would throw the prototype away once it had served its purpose. The prototype's output — FundsXML files for the Europa Growth Fund with real data from February and March 2026 — was the primary input to the testing phase that followed.

13.4.2 The Prototype Architecture

A prototype does not need to be architecturally pretty. Europa's prototype was a single Python script, roughly six hundred lines, that:

1. Connected to the PMS, CRD, and RRDM databases through direct SQL queries.
2. Read the EMT Excel workbook for the regulatory template data.
3. Applied the mapping transformations from §13.3.
4. Built a FundsXML document using `lxml`'s `ElementTree` API.

5. Validated the result against `FundsXML4.xsd` using `xmllint`.
6. Wrote the file to a local directory.

Six hundred lines is not a lot of code, and the prototype worked end-to-end within the first two weeks. The remaining six weeks were spent discovering the *problems* — the things the mapping table did not anticipate, the quirks of the source data, the ambiguous fields where the schema allowed several interpretations and the team had to pick one. Every problem was logged, every fix went into the mapping table or into a "design decisions" document, and every fix was re-tested before the next problem was attacked.

13.4.3 The Problems the Prototype Surfaced

Europa's prototype surfaced more than thirty distinct problems during its six weeks of iteration. Most were small; a few were structural enough to change the project plan. Four are worth describing concretely:

Problem 1 — Currency mismatches between share classes. The prototype produced EMT blocks for each share class, and the EMT `FinancialInstrument_Currency` field should match the share class's own currency. For the R-CHF-ACC-HEDGED class, the team had initially written `EUR` (the fund's base currency) because that was what the first version of the mapping table said. The prototype's validation caught it on the second day: the schema does not check the consistency, but the downstream consumer expected CHF for a CHF-denominated class. The fix was a mapping-table correction: `EMT_FinancialInstrument_Currency` should come from the share class, not from the fund. Fifteen minutes of work; six weeks' worth of potential confusion if it had been found in production.

Problem 2 — FX rate sourcing for TNAV aggregation. The fund-level `TotalNetAssetValue` is the sum of the per-share-class TNAVs after conversion to the fund base currency. The prototype initially used the PMS's own FX-rate table, but the administrator had a *different* FX-rate source (a vendor feed that was updated at the valuation point), and the two rates differed by basis points. The consumer-facing TNAV had to match the administrator's official figure; the pipeline had to switch to the administrator's FX source. The switch required adding a new data feed that the original mapping had not identified.

Problem 3 — Portfolio position ordering. The prototype emitted positions in the order they came out of the PMS database, which was roughly the order of trade dates. The validator passed, but the consumer's downstream comparison tool (which diff'd each month's delivery against the previous month's to flag significant changes) flagged thousands of "positions moved" diffs, because the order had shifted. The fix was to sort positions by ISIN before emission — a trivial change, but only visible once a real consumer's downstream tool had run against the output.

Problem 4 — The CustomAttributes question. Two pieces of producer-specific data — the sequential delivery number and the source-system version that generated the file — had no natural home in the FundsXML schema. The initial mapping had left them unmapped. The prototype team, after consulting Chapter 9, added them as `CustomAttributes` entries in `ControlData` with a namespace-like `eam.delivery.*` prefix. The decision was recorded in the design-decisions document as a permanent project convention.

13.4.4 The Pilot Delivery

Toward the end of the prototyping phase, on 28 April 2026, the team emitted their first **pilot delivery**: a complete FundsXML file for the Europa Growth Fund's 31 March 2026 valuation, containing `ControlData`, `Fund`, `FundStaticData`, `FundDynamicData`, `Portfolios`, `RegulatoryReportings` (EMT, EPT, EET, TPT), `AssetMasterData`, and `Documents`. The file was 14 megabytes, passed schema validation, passed the Schematron rule set the team had

written in parallel, and was sent through the SFTP channel to two cooperative pilot consumers: BNP Distribution Services in France and Comdirect Bank in Germany. Both consumers received the file, parsed it successfully, and reported back any issues they found.

The pilot revealed two further problems, both on the consumer side. BNP reported that their PRIIPs KID generator expected the `EPT_FinancialInstrument_UmbrellaName` field to be populated, but the Europa Growth Fund (modelled in the book and in the prototype as a standalone fund) had none. The fix was to set the umbrella name to a placeholder `"Europa Asset Management Investments"` — the umbrella name the fund *would* have if it were modelled correctly as a sub-fund — and to treat the real migration to an umbrella structure as a follow-on task. Comdirect reported a purely cosmetic issue: their ingestion was strict about the XML declaration's `encoding` attribute being `UTF-8` exactly, and the prototype had omitted the encoding attribute entirely. The fix took ten minutes.

Both consumer issues were absorbed before the prototype phase ended, and the delivery was re-sent on 29 April. The second pilot was accepted by both consumers without further issues. The prototype had done its job: the team now had a concrete, working example of every data flow, every transformation, every validation, and every consumer interaction. They were ready to start the production build.

13.5 Phase 4: Testing

13.5.1 What Testing Means for a FundsXML Project

With the prototype retired and the production code in development, the testing phase (1 May to 31 May 2026) focused on verifying that the production implementation matched the prototype's behaviour at scale and under all the edge cases the prototype had not seen. The testing phase had three layers.

Layer 1 — Unit tests of each transformation step. Each mapping rule from §13.3 became a unit test: given a known source row, the transformation must produce a known target value. Europa's unit-test suite grew to roughly 400 tests by the end of the month, each independently runnable, each taking milliseconds, each giving a clean pass/fail that told the developer immediately when a change had broken a rule.

Layer 2 — Integration tests that run the whole pipeline end-to-end against a fixture dataset. The fixture was a snapshot of the source databases from 31 March 2026 (the valuation date the prototype had used), with deliberate modifications to exercise edge cases: missing optional fields, a share class with no portfolio positions (a theoretical case that should not happen but might), a late-arriving CORRECTION that replaces an earlier delivery, a day with no subscriptions or redemptions, a day with a single very large subscription. The integration-test suite ran roughly a dozen scenarios, each producing a FundsXML file, each validated against XSD and Schematron, each compared against a golden-output fixture.

Layer 3 — Acceptance tests with the pilot consumers. Real FundsXML files produced from real source data, sent through real delivery channels, consumed by real consumer systems. BNP, Comdirect, and two further distributors were involved in this layer. The acceptance tests ran three deliveries: a normal month-end file, a correction of a deliberately wrong earlier file, and a delete of a file that should never have been sent. All three exercised the full producer-consumer handshake and exposed any remaining issues.

13.5.2 The Test Fixtures

A good fixture is the foundation of effective testing. Europa's fixtures came from three sources:

- **Real historical data.** Four months of source-database snapshots (December 2025 through March 2026) were archived in a dedicated fixture repository and used as input to the integration tests. Real data is the best test data because it exercises the quirks and edge cases the team cannot anticipate.
- **Synthetic edge cases.** For scenarios that real data did not cover — a SPLIT corporate action, a share class with zero shares outstanding, a day with a negative net flow — the team hand-crafted fixtures by taking a real snapshot and modifying specific fields. Synthetic fixtures are the right tool for exercising error paths and edge cases that rarely occur in production.
- **Generated samples.** For structural tests that needed complete schema coverage (every optional element populated at least once), the team used the FundsXML Generator from Chapter 11 to produce comprehensive fixtures. Generated fixtures are the right tool for structural completeness but the wrong tool for business-logic correctness, because the generated values are plausible but not semantically meaningful.

Europa's fixture repository held roughly seventy distinct fixtures by the end of the testing phase, organised into categories (normal, edge-case, error-case, regression). Each fixture had a short README describing its intent, so that a new engineer encountering the repository six months later could understand what each one was for.

13.5.3 The Bugs Found in Testing

Testing found twenty-two bugs that the prototype had missed, most of them small. Four are worth describing because they illustrate typical categories:

Bug 1 — Integer overflow on a share count. A test fixture with 2.1 billion shares outstanding (deliberately set higher than the PMS's normal maximum) overflowed a 32-bit signed integer in the transformation layer. The fix was to use a 64-bit type for the share-count field. The bug had been dormant in the prototype because no real data had ever approached the overflow threshold; the fixture was specifically designed to exercise it.

Bug 2 — Missing `RelatedDocumentIDs` on a generated AMEND. The pipeline produced an AMEND delivery in the correction-test scenario, but forgot to populate the `RelatedDocumentIDs` element pointing at the original delivery. The Schematron rule from Chapter 10 caught it on the next validation run. The fix was to add the back-pointer to the producer's generation logic; the test case was retained as a permanent regression fixture.

Bug 3 — Locale-dependent number formatting. The production pipeline, deployed on a Linux container configured for the German locale, emitted decimal numbers with commas as the decimal separator (`124,5078`) instead of the English ISO-standard period (`124.5078`). The bug manifested only in production configuration, not in the unit tests, which ran under the default English locale. The fix was to force the JVM locale to `en-GB` (the British English locale that uses period-as-decimal and is the closest to "ISO English" for number formatting purposes) at startup. This kind of locale dependency is a classic production-only bug.

Bug 4 — File-not-found on a concurrent run. A test scenario that triggered two pipeline invocations simultaneously revealed that both invocations were writing to the same temporary file, and one of them was corrupting the other's output. The fix was to add a per-invocation UUID to the temporary file names. The bug was uncovered by a chaos-test that deliberately triggered overlap; in production, the scheduler would normally prevent overlap, but the defensive fix removed the risk entirely.

13.5.4 Test-Driven Scope Control

An important operational discipline during testing: when a bug was found, the team first wrote a **failing test fixture** that reproduced the bug, then fixed the code, then verified that the test passed. The fixture was retained permanently in the regression suite. This discipline had two benefits: every bug became a permanent test case that would catch the same bug if it recurred, and the fixture repository grew organically to cover the real edge cases that mattered. By the end of testing, the fixture repository was the institutional knowledge of "things that have gone wrong with this pipeline", and it remained the most valuable testing asset through the project's go-live and into operations.

13.6 Phase 5: Acceptance and Go-Live

13.6.1 Formal Acceptance

User acceptance testing (UAT) ran from 1 June to 28 June 2026, with stakeholders from Operations, Fund Administration, Compliance, and the six pilot distributors participating. The UAT process was straightforward: the pipeline produced a set of deliveries, each stakeholder reviewed the output relevant to their role, and each stakeholder signed off that the output met their expectations. Failures during UAT went back to the development team for fixing and re-testing.

Three UAT sign-offs mattered most.

Operations sign-off confirmed that the pipeline could be operated by the existing Fund Operations IT team after go-live — that the runbook was clear, the monitoring dashboards were understandable, the alerting thresholds were appropriate, and the runbook actions for each likely failure mode were achievable by on-call staff without needing to escalate to the development team. Operations signed off on 18 June after two rounds of runbook revisions.

Compliance sign-off confirmed that the regulatory modules (EMT, EPT, EET, TPT) were populated correctly and that the audit trail was sufficient to defend the delivery chain in a regulatory inspection. Compliance signed off on 22 June.

Distributor sign-off was the most operationally important. Each of the six pilot distributors received two deliveries during UAT and confirmed that their own ingestion pipeline processed them correctly, that the resulting downstream fact sheets and regulatory documents were correct, and that they were ready to receive production deliveries. Five of the six signed off by 24 June; the sixth (one of the smaller distributors) requested a follow-up adjustment to the `DocumentURL` scheme in the Documents section and signed off on 27 June, one day before the UAT deadline.

13.6.2 Go-Live Strategy — Parallel Run

The go-live strategy was **parallel run**: for the month of July 2026, both the legacy CSV pipeline and the new FundsXML pipeline would emit deliveries for the same valuation dates. Each distributor would process whichever format their own systems preferred. Europa's operations team would compare the two outputs daily and investigate any discrepancies. This approach was slower and more expensive than a hard cutover, but it guaranteed that the legacy pipeline was still available as a safety net if the new pipeline failed.

The first parallel-run delivery was emitted on 1 July 2026 at 06:47 UTC for the 30 June 2026 valuation point — the first production FundsXML delivery in Europa Asset Management's history. Both pipelines produced their outputs, both passed their respective validations, and both reached all six distributors within the operational window. Five distributors confirmed successful ingestion of the FundsXML version by 09:00 UTC; the sixth confirmed by 11:30 after a brief issue with its SFTP configuration. Day 1 was a success.

13.6.3 The First Month in Production

The first month of parallel run surfaced three operational issues. None were severe enough to trigger a rollback, but each required a follow-up.

- **An SFTP timeout on day 3.** One of the distributors' SFTP drop-boxes experienced a brief outage, and the pipeline's retry logic was more conservative than the distributor's recovery time. The pipeline had already failed the delivery and alerted when the distributor recovered. Operations re-emitted the delivery manually. The fix was to extend the retry timeout; rolled out on day 6.
- **A locale issue on day 8.** The locale bug from §13.5.3 reappeared in a slightly different form — the German locale configuration had been set correctly on the production server, but one of the regulatory modules was using a separate formatting library that picked up the system locale independently. The fix was to force the locale on that library explicitly. Rolled out on day 9.
- **A performance issue on day 17.** The month-end delivery took 45 minutes to produce, when the pipeline's design target had been 20 minutes. Investigation found that a database query against the portfolio table was doing a full table scan instead of using an index that the test environment had had but the production database did not. The fix was to add the missing index. Rolled out on day 18.

By the end of July, all three issues were fixed, the parallel run had completed successfully for every daily and month-end delivery, and the decommissioning of the legacy CSV pipeline could be scheduled for 31 August 2026. Europa Asset Management had successfully transitioned to FundsXML as its authoritative fund-data format.

13.7 Phase 6: Operations and Maintenance

13.7.1 The Day After Go-Live

Go-live is the end of the project, but it is the beginning of the operation. The project team that built the pipeline typically disbands within a few weeks of successful go-live, and the system passes into the care of a permanent operations team that is — by definition — less familiar with the code than the builders were. The transition from "project" to "operations" is one of the riskiest moments in a FundsXML implementation, because institutional knowledge evaporates quickly if it is not captured deliberately.

Europa's transition plan had four elements.

A runbook documenting every production procedure: how to start and stop the pipeline, how to re-emit a failed delivery, how to investigate a validation failure, how to rotate credentials, how to roll back to the legacy pipeline in an emergency. The runbook was written during the testing phase (not after go-live, when the team would have less

patience for it) and was the primary hand-off artefact to operations.

A dashboard showing the health of the pipeline at a glance: last successful delivery, last failed delivery, queue depth, average validation time, per-distributor ingestion-confirmation status. The dashboard was built in the company's existing Grafana instance (no new infrastructure) and was the first thing the on-call engineer looked at every morning.

An alerting policy defining which conditions trigger a page to on-call staff and which do not. Europa's policy: failed delivery within the emission window pages immediately; failed ingestion confirmation from a distributor pages within 30 minutes; delayed delivery (more than 30 minutes behind schedule) pages within the emission window but not outside it; validation warnings (non-blocking) are logged but do not page.

A knowledge transfer plan of sessions between the project team and the operations team, held during the last three weeks of the project. The sessions walked through the pipeline's architecture, the fixture repository, the runbook procedures, and the most likely failure modes. The goal was that by the time the project team disbanded, the operations team could handle every Tuesday-morning failure without calling the developers.

13.7.2 The First Six Months of Operations

By January 2027 — six months after go-live — Europa's FundsXML pipeline had emitted approximately 180 production deliveries (daily NAVs for the retail classes plus month-end deliveries plus ad-hoc regulatory deliveries) without a single emission that had caused downstream incident at a distributor. The operational metrics were:

- **Availability:** 99.8% of scheduled deliveries made it out within the emission window (against a 99.5% target).
- **Validation pass rate:** 100% of emitted deliveries passed both XSD and Schematron validation before emission (no bad delivery reached a distributor).
- **Incident count:** eleven operational incidents, all resolved within the on-call response SLA, none causing regulatory consequences.
- **Schema-upgrade handling:** one — FundsXML 4.2.9 was released by the FundsXML association in November 2026, and Europa's pipeline needed to be updated to match. The upgrade took two developer-weeks and was deployed without incident.

The operations team had absorbed the pipeline as an ordinary part of its workload. The project team had disbanded in late August 2026 with a handful of follow-on tasks (the multi-language text build-out, the umbrella-structure migration, the data-gap backfill) that were scheduled into the normal engineering backlog rather than treated as urgent.

13.7.3 Schema Upgrades — Continuous Maintenance

FundsXML releases minor updates two or three times a year, and each release can introduce new fields, deprecate old ones, or tighten validation rules. A producer that ignores these updates will eventually emit files that consumers reject because they are built against an older schema than the consumer expects. Staying current is therefore not optional — it is a continuous maintenance task.

Europa's schema-upgrade process, refined after the 4.2.9 upgrade, has five steps:

1. **Monitor** the FundsXML release channel (the GitHub releases page, the official mailing list). New releases are typically announced four to six weeks before they become mandatory for consumers that want to use the new features.
2. **Review** the changelog. Every minor release has a changelog enumerating the fields added, removed, or changed. Most fields are backwards-compatible additions that do not affect the producer's output; occasionally a release requires the producer to emit a new mandatory field.
3. **Update** the pipeline's schema reference (FundsXML4.xsd), update any mapping-table entries affected by the change, and run the test suite against the new schema.
4. **Pilot** the new version with one distributor before rolling out to all. A pilot reveals any consumer-side surprises that the test suite does not catch.
5. **Roll out** to all distributors, coordinated with the consumer side where possible.

The full cycle takes two to four developer-weeks of work per release, depending on the scope of the changes. A mature producer typically has this workflow documented and practised enough that it becomes an ordinary part of the engineering cadence rather than a disruptive event.

13.7.4 Producer-Consumer Relationship as an Ongoing Conversation

One operational lesson from Europa's first six months deserves to be stated explicitly: **the producer-consumer relationship is not a one-time contract signed at go-live; it is an ongoing conversation.**

Distributors occasionally request changes to the data they receive — a new field they need for a new regulatory disclosure, a different format for a cost figure, an additional language in the description text. Regulators occasionally update the rules that drive the templates — EMT, EPT, EET, TPT all evolve on their own release cadences. Internal data sources change, not always in ways that are visible from the pipeline's perspective. The producer's operations team has to monitor the conversation, route requests to the right responders, and occasionally push back when a requested change would be disruptive.

Europa's approach, again refined after six months, is to hold a **quarterly review meeting** with each distributor relationship, in which both sides share feedback on the recent deliveries and flag any upcoming needs. The meetings are short (30 minutes per distributor), informal, and structured around three questions: "Is everything working?" / "Is anything about to change on your side?" / "Is anything about to change on our side?" The meetings replace the need for ad-hoc firefighting when a change surprises one side; they catch the surprises early and let both sides plan together.

13.8 The Europa Growth Fund Timeline — A Case-Study Recap

The project described in this chapter ran from December 2025 to August 2026 — nine months from kickoff to go-live, followed by ongoing operations. The timeline, in summary:

Table 13.2 — The Europa Asset Management FundsXML project timeline

Phase	Dates	Duration	Key deliverables
Phase 0 — Trigger and kickoff	Oct–Dec 2025	2 months	Audit finding, distributor notice, project charter
Phase 1 — Requirements analysis	8 Dec 2025 – 31 Jan 2026	2 months	Requirements document, stakeholder sign-off
Phase 2 — Data mapping	1 Feb – 28 Feb 2026	1 month	Mapping table, data-gap list, quality-issue list
Phase 3 — Prototyping and piloting	1 Mar – 30 Apr 2026	2 months	Working prototype, two pilot deliveries accepted
Phase 4 — Testing	1 May – 31 May 2026	1 month	Unit/integration/acceptance test suites, regression fixtures
Phase 5 — UAT and go-live preparation	1 Jun – 30 Jun 2026	1 month	Stakeholder sign-offs, runbook, dashboards, alerting
Phase 6a — Parallel run	1 Jul – 31 Jul 2026	1 month	First production deliveries, legacy comparison
Phase 6b — Legacy decommission	31 Aug 2026	—	Legacy CSV pipeline retired
Phase 6c — Steady-state operations	Sep 2026 onwards	Indefinite	Ongoing deliveries, schema upgrades, relationship reviews

Several observations about the timeline are worth making.

Requirements and mapping together consumed three months — one-third of the project duration. Newcomers to FundsXML implementation projects typically underestimate these phases and overestimate the build phases; the Europa project's distribution of effort is more realistic than the typical newcomer plan.

Prototyping consumed two months — twice the one month the initial plan had allocated. The team had pre-negotiated an additional slack week per phase, and the prototyping phase consumed all of it plus a week of the testing phase. This is typical: the first phase where code meets data is almost always longer than planned.

Testing consumed the allocated month exactly. The unit, integration, and acceptance tests all fit into their slots, largely because the prototype had already surfaced the most time-consuming problems.

UAT consumed the allocated month exactly as well, with one late sign-off from the smallest distributor. The parallel run and the go-live went smoothly, though not without three operational issues in the first month that the team had to address quickly.

The post-go-live schema upgrade happened on schedule (FundsXML 4.2.9 in November 2026) and cost two developer-weeks — a small but recurring maintenance cost that was comfortably absorbed by the operations team's normal workload.

Nine months from kickoff to production is a realistic timeframe for a first-time FundsXML implementation at a mid-sized asset manager with a clean scope, a cooperative set of distributors, and a pragmatic team. Larger organisations with more internal stakeholders, or messier source data, or more complex distribution relationships, typically take twelve to eighteen months for the same scope. The Europa project was successful partly because the scope was kept tight, the team was cross-functional from day one, and the stakeholders stayed engaged through the full timeline.

13.9 Common Pitfalls

- **Underestimating the mapping phase.** Every newcomer project thinks mapping will take a week; every mature project knows it takes a month. The mapping is where the source data's quirks become visible, and those quirks are the hardest problem the project will face. Plan for a full phase on mapping, not a week.
 - **Skipping the prototype.** Building the production pipeline directly, without a throwaway prototype, causes the production code to absorb every early mistake as permanent technical debt. The prototype is cheap; the production code bears the cost of its mistakes for years.
 - **Ignoring the pilot consumer.** A pilot delivery to a real consumer surfaces problems that no amount of internal testing can find, because only a real consumer's downstream tools exercise the full integration. Every project should have at least one cooperative pilot consumer engaged from the prototyping phase onwards.
 - **Treating go-live as the end.** Projects that celebrate go-live and then disband the team leave the operations team without institutional knowledge and the pipeline without a mechanism for ongoing maintenance. The transition from project to operations is a deliberate phase, not an afterthought.
 - **Cutting over hard instead of running in parallel.** A hard cutover gives the project a romantic "all-in, no turning back" feeling, but it leaves no safety net when something goes wrong. A parallel run for at least one month is the professional standard, and the cost (running both pipelines for a few weeks) is much lower than the cost of a roll-back from a failed hard cutover.
 - **Ignoring schema upgrades.** FundsXML releases minor updates several times a year, and each one needs to be absorbed by the pipeline. A producer that lets the schema drift will eventually emit files that consumers reject; the maintenance cost is permanent but small if treated as part of the engineering cadence.
 - **Letting the producer-consumer relationship go stale.** The relationship is an ongoing conversation, not a one-time contract. Regular check-ins (quarterly, even informally) prevent ad-hoc firefighting when one side's needs change.
 - **Building the pipeline without involving Compliance.** Compliance sign-off is not optional, and late Compliance engagement typically surfaces regulatory gaps that force painful rework. Engage Compliance from the requirements phase onwards, and keep them involved through every gate.
-

13.10 Key Takeaways

- A FundsXML implementation project at a mid-sized asset manager typically runs **nine to twelve months** from kickoff to first production delivery, through six distinct phases: requirements, mapping, prototyping, testing, acceptance and go-live, and operations.
- **Phase 1 (requirements)** produces a tight scope document naming the funds, share classes, regulatory modules, delivery channels, and — explicitly — the things the project will *not* do. The out-of-scope list is as important as the in-scope list.
- **Phase 2 (mapping)** is the hardest phase. It produces a mapping table with one row per FundsXML field, naming the source system, source column, transformation, and default for each. The mapping exercise surfaces data gaps and data-quality issues that the project must address before production.
- **Phase 3 (prototyping)** builds a throwaway implementation in a fast language (Python, typically) to discover the problems that only become visible when code meets data. A pilot delivery to a cooperative real consumer at the end of the prototype phase surfaces further integration problems that internal testing alone cannot find.
- **Phase 4 (testing)** has three layers — unit tests per mapping rule, integration tests end-to-end against fixtures, acceptance tests with real consumers. Every bug found becomes a permanent regression fixture, and the fixture repository is the institutional knowledge of the pipeline's edge cases.
- **Phase 5 (UAT and go-live)** collects formal sign-offs from Operations, Compliance, and each pilot distributor, then runs the new pipeline in parallel with the legacy pipeline for at least a month before the legacy is decommissioned. Parallel run is the professional standard; hard cutover is a gamble.
- **Phase 6 (operations)** begins at go-live and continues indefinitely. It requires a runbook, a dashboard, an alerting policy, and deliberate knowledge transfer from the project team to the operations team. Schema upgrades are a recurring maintenance task (two to four developer-weeks per FundsXML minor release), and the producer-consumer relationship is an ongoing conversation that deserves regular check-ins.
- The Europa Asset Management project that forms this chapter's narrative ran from December 2025 to August 2026 and is a realistic template for a first-time implementation at a mid-sized asset manager. It succeeded because the scope stayed tight, the team was cross-functional, and the stakeholders stayed engaged through the full timeline.

Chapter 14 looks forward from this operational baseline to where FundsXML is headed — the current development directions of the standard, its role in an API-driven world, the influence of AI and automation, and the ways readers can contribute to the community that shapes the schema.



PART IV — OUTLOOK AND REFERENCE · CHAPTER 14

Future and Further Development

Where is FundsXML heading?

14.1 Setting the Scene: Standing on Thirteen Chapters

Thirteen chapters ago, we began with a single question: what does it actually mean to have a European standard for fund data exchange, and why should anybody care? The answer has taken a book to unfold. We introduced the problem space and the ecosystem of producers and consumers (Part I). We worked through the schema itself, element by element, from [ControlData](#) through Funds, Portfolios, Transactions, and the FinDatEx regulatory modules (Part II). We moved into the practical questions of validation, tooling, system integration, and implementation projects (Part III). What remains is the final question — and it is the question that matters most if FundsXML is to stay useful beyond 2026: **where is it going next?**

This is a deliberately shorter and more reflective chapter than the ones that came before it. It contains no schema listings, no code, no XML. What it offers instead is a careful look at four intersecting currents: the ongoing evolution of the standard itself, the tension between XML and API-driven architectures, the role that AI and automation are beginning to play in the working life of a data engineer, and — perhaps most importantly — the ways in which you, the reader, can become part of the community that shapes what FundsXML looks like in 2028 and beyond.

A word of honesty before we start. Predictions about the future of any technology standard are always at least partially wrong. Some of what this chapter describes as "likely directions" will turn out to be dead ends; other developments that nobody is talking about in early 2026 will dominate the conversation five years from now. The chapter is therefore framed less as a forecast and more as a compass: these are the directions the community is pointing, these are the forces pulling the standard one way or another, and these are the places where a thoughtful practitioner can pay attention, contribute, and — when necessary — push back. Treat it as an invitation rather than an oracle.

By the end of this chapter, you should be able to:

- describe the current development directions of the FundsXML standard as of early 2026;
 - weigh the practical trade-offs between XML-native pipelines and JSON/API facades, and decide where each fits;
 - understand the role that AI and automation can realistically play in FundsXML work — and the places where they should not yet be trusted;
 - name concrete ways in which practitioners can contribute to the FundsXML community, from filing a GitHub issue to joining a working group;
 - take the book's thirteen chapters of technical material and turn them into active engagement with the standard rather than passive consumption.
-

14.2 The FundsXML Standard in Motion

14.2.1 A Standard Is Never Finished

The FundsXML schema has been in continuous evolution since its first public release two decades ago, and it will continue to evolve. The current major version, 4.2, has been stable in its overall shape since 2019, but its minor releases — 4.2.0 through 4.2.8 as of early 2026, with 4.2.9 anticipated in November 2026 — have each added fields, tightened validation rules, or reorganised a piece of the schema in response to a specific community request. A producer or consumer that treats the schema as a finished artefact and freezes its implementation against a single minor release will, within two years, be running against a version that the rest of the ecosystem has already moved past.

The rhythm of releases is not arbitrary. The FundsXML association, the non-profit body that stewards the standard, maintains a published change process and a release cadence of roughly two to three minor releases per year. Each release is preceded by a consultation period during which proposed changes circulate through the member community for review, revision, and — occasionally — rejection. The process is slower than a single-vendor product roadmap but faster than most ISO or ESMA standards, and the balance has served the community well enough that the schema has not undergone a disruptive major-version transition since the move from 3.x to 4.0 many years ago.

14.2.2 What the 2026 Roadmap Is Thinking About

The topics under active discussion in the FundsXML working groups as of early 2026 cluster around four themes. None of them is guaranteed to become part of a future release, and the specific shape each takes in the schema will depend on the community's decisions over the coming months and years. But the themes themselves are stable enough to describe.

Digital and tokenised assets. The emergence of tokenised funds — funds whose shares are issued as blockchain-based tokens rather than as traditional book-entry units — raises several questions that the current schema does not answer cleanly. A tokenised share class still has a LEI, a currency, a NAV, and regulatory disclosures, so most of the existing FundsXML structure applies. But it also has attributes that the existing schema has no vocabulary for: the smart-contract address that represents it, the blockchain network it lives on, the distinction between on-chain and off-chain custody, the treatment of gas fees in the cost disclosures. Whether these are added as extensions to existing elements, as a new sub-block, or via the CustomAttributes extension mechanism is an open design question. The community has not yet settled on a direction, but the conversation has begun.

ESG and sustainability reporting extensions. The SFDR framework has been evolving since 2021, and each iteration has added data fields that the EET template must carry. Current discussions around SFDR Level 3 and around the EU Taxonomy regulation point toward further extensions to the EET module — additional PAI indicators, more granular Taxonomy alignment reporting, and possibly a new section for biodiversity-related disclosures. FundsXML's flexible structure handles these additions well, but each one requires careful negotiation about backward compatibility with earlier EET versions. A producer that emits EET v1.1.2 in January 2026 will probably need to emit a new version by January 2028, and the transition planning should already be on the roadmap.

T+1 and settlement acceleration. The US equities market moved to T+1 settlement in May 2024, and the European Commission has signalled that European markets will follow in 2027. Faster settlement changes the operational rhythm of fund accounting, because the window between trade date and the next NAV calculation tightens, and in some cases the NAV publication schedule itself shifts. FundsXML's dynamic-data elements already carry trade and

settlement dates, so the schema does not need structural changes to support T+1, but the *practical* implications for producers are significant: validation and emission pipelines that are comfortable with the T+2 cadence will need to be tested against the tighter T+1 schedule, and some of the assumptions baked into existing schedulers may need to be revisited. The association is tracking the topic and may issue guidance — though not schema changes — as the 2027 date approaches.

Cross-border distribution under the new UCITS framework. The UCITS directive is itself under quiet revision, with a working document circulating in Brussels that proposes clarifications around the cross-border distribution notification process. If the revisions land in their current form, they will require additional fields in the distribution-country block — perhaps a structured notification-status indicator, or an explicit distribution-authorisation date per country. These would be small additions but would need to be coordinated with the consumer side (the national competent authorities) so that the new fields are consumed as well as produced.

Beyond these four, smaller discussions touch on improved support for master-feeder structures, clearer conventions for funds-of-funds with multiple layers, and an ongoing cleanup of deprecated elements from the 3.x era that are still technically present in 4.2 for compatibility but are no longer used by any active producer. None of these will reshape the schema dramatically, but each one is a piece of the continuous maintenance that keeps the standard usable.

14.2.3 Governance Through the FundsXML Association

The FundsXML association itself is worth describing briefly, because its governance model is what makes the continuous-evolution cadence possible. The association is registered in Austria, funded by membership fees from its corporate members (asset managers, fund administrators, custodians, technology vendors, and distributors), and steered by a board elected from the membership. The day-to-day work — the working groups, the consultation periods, the release coordination — is done partly by salaried staff and partly by volunteers seconded from member organisations.

The consequence of this model is that FundsXML is neither a vendor product (with the risks of commercial capture) nor a regulator-owned standard (with the risks of slow adoption and political compromise). It sits in the middle: a community-owned standard, stewarded by the people who use it day-to-day, with enough formal governance to prevent chaos and enough informality to move faster than a top-down process would permit. The model has worked for twenty years, and there is no reason to think it will not continue to work for another twenty — provided the community remains engaged.

14.2.4 The Relationship with FinDatEx

A particular feature of the 2020s has been the growing relationship between the FundsXML association and FinDatEx, the European Working Group on Investment Data Exchange. FinDatEx owns the five regulatory templates that Chapter 8 covered in detail — EMT, EPT, EET, EFT, TPT — and it publishes them under its own governance. FundsXML embeds those templates in its [RegulatoryReportings](#) block, which means that every time FinDatEx publishes a new EMT or EET version, the FundsXML schema has to be updated to embed the new version, and producers have to coordinate their upgrades to match.

The two organisations have developed a productive working rhythm: FinDatEx leads on the content of the regulatory templates (which fields, which values, which enumerations), and the FundsXML association leads on the wrapper structure and on the integration with the rest of the fund-data schema. Coordination meetings happen several times a

year, and the release calendars of the two organisations are increasingly aligned so that a producer can plan a single upgrade cycle that handles both the FinDatEx template changes and the FundsXML schema changes together. This kind of cross-organisational cooperation is difficult to achieve and easy to lose; the fact that it is working well is one of the quieter success stories of the European fund-data ecosystem.

14.3 FundsXML in an API-Driven World

14.3.1 The XML-Versus-JSON Conversation, Honestly

No chapter about the future of an XML-based standard can avoid the question that newcomers always ask: shouldn't FundsXML just become JSON? The question deserves an honest answer, because the reflexive defences ("XML is more expressive", "JSON lacks a proper schema language") are only partly true and do not settle the argument by themselves.

The honest answer has three parts.

First: **FundsXML's core value is not the serialisation format, it is the data model.** The hundreds of precisely defined elements, the enumerations, the cross-references between funds and portfolios and regulatory modules — all of that is what makes FundsXML useful, and all of it could in principle be expressed in JSON, YAML, Protocol Buffers, or any other structured format. The XML syntax is a historical choice that made sense in the early 2000s and remains a reasonable choice in 2026 for the file-based batch-oriented workflows that dominate fund data exchange. But if a future version of the standard were to offer a canonical JSON serialisation alongside the XML one, the core model would survive the transition unchanged.

Second: **XML and JSON are good at different things, and FundsXML's workload leans XML.** XML's strengths — hierarchical documents with mixed content, formal schema validation, in-document digital signatures, rich namespace mechanics, mature tooling for transformation (XSLT) and query (XPath, XQuery) — are exactly the strengths that a regulatory disclosure document needs. JSON's strengths — compactness, JavaScript-native parsing, easy REST API integration, and a more minimalistic philosophy — are the strengths that a real-time API needs. The two do not compete so much as they cover different domains. A fund administrator's monthly batch delivery is an XML-shaped problem; a mobile app querying for a single fund's current NAV is a JSON-shaped problem. Both can be true at the same time.

Third: **the future probably includes both.** A likely evolution over the next several years is that FundsXML remains the authoritative specification and the canonical batch format, while a complementary JSON projection emerges — not as a replacement but as a companion. The JSON projection would serve API-driven use cases (real-time queries, mobile clients, lightweight integrations) while the XML form continues to handle regulatory batch delivery. The two would describe the same underlying data model and would be kept in sync by automated transformation, so that a producer could emit either form from a single pipeline and a consumer could read either form against the same business logic.

14.3.2 REST and GraphQL Facades over FundsXML Pipelines

In practice, several producers and consumers have already built REST or GraphQL facades in front of their FundsXML pipelines, and the pattern is worth describing because it lets teams get the API-driven experience without abandoning the XML model.

The typical architecture is this: the producer maintains its normal FundsXML pipeline as the authoritative source of truth, emitting monthly batch files exactly as Chapter 12 described. Alongside the batch pipeline, a small web service exposes a REST endpoint that, on request, returns a JSON projection of the most recent FundsXML delivery for a given fund. The service reads the XML file, extracts the requested subset, converts it to JSON, and returns it. No new source of data is created; the XML file remains the authoritative record, and the JSON is a view.

The benefits are concrete. A mobile app querying fund NAVs no longer needs to parse a 15-megabyte XML file for a single number; it gets a 200-byte JSON response. A dashboard displaying portfolio breakdowns can query the API and render the data without understanding the FundsXML schema at all. Third-party integrations — fintech aggregators, robo-advisors, financial planning tools — can consume the API with standard JSON tooling and never encounter XML at all. The producer gets to serve both the regulatory-batch consumer and the API-driven consumer from a single source.

The costs are also real. The REST facade is additional infrastructure to build and operate, and it inherits the problem of cache invalidation — when a new FundsXML delivery arrives, any cached API response becomes stale, and the invalidation logic has to be correct. The facade also tends to expose only a subset of the schema (the subset the API designers thought was useful), which limits its utility for consumers that need fields the API did not expose. And the facade is a new surface area for authentication, authorisation, and access logging, each of which has to be handled correctly.

For the Europa Growth Fund, the implementation project described in Chapter 13 did not build a REST facade, because the six distributors all preferred batch SFTP delivery. But a seventh distributor — a French robo-advisor platform that onboarded Europa as a fund in its retail catalogue in late 2026 — requested a REST endpoint for real-time NAV queries, and the Europa operations team built one in early 2027. The facade was approximately 400 lines of Python, took two developer-weeks to build, and has been running comfortably since. It does not replace the batch pipeline; it complements it.

14.3.3 The European Single Access Point

A different kind of API conversation is happening at the regulatory level, in the form of **ESAP — the European Single Access Point**. ESAP, established by Regulation (EU) 2023/2859 and scheduled to become operational in 2027, is a centralised platform that aggregates regulatory disclosures from all EU-listed entities and makes them publicly searchable. Funds are within ESAP's scope, and the fund information that flows through ESAP is structured data — the kind of structured data that FundsXML is built to carry.

The exact relationship between ESAP and FundsXML is still being worked out as of early 2026. The most likely outcome is that ESAP accepts fund disclosures in whatever format the producer already uses — FundsXML for producers that already use it, other formats for producers that do not — and normalises them internally into a searchable index. The FundsXML association is engaged with the ESAP working group to ensure that FundsXML-native submissions are well-supported and that the normalisation does not lose information that the schema carries. If this goes

well, ESAP becomes one more consumer in every producer's consumer list, with a well-defined ingestion channel and a clear value proposition for the end users (analysts, journalists, regulators, academics) who benefit from centralised access to fund data.

The broader lesson from the ESAP discussion is that FundsXML's role is shifting from *an industry-internal format* to *a public-good format*. Ten years ago, a FundsXML delivery went from one bank to another and was seen by a handful of downstream systems. In 2027, the same delivery may also flow into a public-access platform where retail investors, journalists, and researchers can query it. This shift changes the calculations around data quality, privacy, and the readability of the schema — and it raises the stakes of getting the community governance right.

14.3.4 Event-Driven and Streaming Architectures

A final architectural direction worth mentioning briefly is the move toward event-driven and streaming architectures for fund data. Some producers and consumers are experimenting with real-time event streams — Kafka topics, AWS Kinesis streams, or similar — where each NAV calculation, each portfolio change, each corporate action is published as an event the moment it happens, rather than aggregated into a batch file at end-of-day. Consumers subscribe to the event stream and process events as they arrive, giving downstream systems near-real-time visibility into fund state changes.

Whether FundsXML has a role to play in streaming architectures is an open question. The schema is designed for document-level deliveries (a complete snapshot of a fund at a valuation point), not for per-event messages, and retrofitting an event-based encoding onto the existing element hierarchy is not straightforward. One possibility is that the batch format remains FundsXML while the event stream uses a simpler per-event schema that is cross-referenced to FundsXML for full context. Another possibility is that the FundsXML schema gains an explicit event-envelope construct in a future version, allowing the same schema to be used for both batch and streaming workloads. The community has not decided which way to go, and it may turn out that neither is necessary — that batch FundsXML and streaming proprietary formats can coexist indefinitely.

14.4 AI, Automation, and the Next Layer of Tooling

14.4.1 What AI Can Realistically Help With

The arrival of large language models as a practical working tool — a development that accelerated dramatically between 2022 and 2026 — has reached the FundsXML community the way it has reached every other technical field. The question every practitioner asks is the same one: where can AI help, and where should it be kept away from the production path?

The honest answer is that AI is already helping in several concrete ways, and the list of useful applications is growing. Three are worth describing in some detail.

Mapping assistance. Chapter 13 described data mapping as the hardest phase of an implementation project, because it requires understanding both the FundsXML schema and the source data well enough to connect them field by field. An LLM that has been trained on the FundsXML schema documentation and given access to a sample of the source

data can produce a *first draft* of a mapping table in minutes — suggesting which FundsXML element each source column probably maps to, flagging ambiguous cases for human review, and generating the transformation rules that the source values probably need. The first draft is never the final mapping; a human engineer still has to review every row, correct the errors, and add the edge cases the model missed. But the first draft turns the mapping phase from "write 450 rows from a blank page" into "review and correct 450 rows that already exist", which is a substantially easier task and measurably faster to complete. Several early adopters have reported mapping phases compressing from six weeks to two or three using this pattern.

Schema migration and upgrade analysis. When a new FundsXML version is released — say, 4.2.9 after 4.2.8 — an LLM can compare the two schema versions and summarise the changes in natural language, flagging which changes will affect a given pipeline based on which fields that pipeline uses. The upgrade impact analysis that would previously have taken a careful human reading of a changelog can be done in a few minutes, with the human reviewer checking the model's conclusions rather than producing them from scratch. This is a low-risk application because the output is a summary for a human to act on, not a direct change to production code.

Schematron rule suggestions. Chapter 10 introduced Schematron as the layer where business rules are enforced beyond what XSD can express. Writing Schematron rules is not difficult, but it is tedious, and the rules that a well-run producer needs can number in the hundreds. An LLM that has been given the FundsXML schema and a description of the business rules in natural language can generate candidate Schematron rules that the engineer then reviews, tests, and adjusts. The generated rules are rarely perfect but are often close enough to be useful starting points, and the overall effort to reach a working rule set drops noticeably.

14.4.2 Where AI Should Not (Yet) Be Trusted

The applications above share a common pattern: **AI generates a first draft, a human reviews and corrects it, the human remains accountable for the result.** This pattern is safe because the human's judgement is in the loop at the point where the work becomes consequential. The pattern breaks down — and the risk becomes real — when AI is used to generate content that then flows directly into production without human review.

Three applications in particular should be kept out of the production path as of 2026, and each deserves an explicit warning.

Generating regulatory module content from unstructured sources. A tempting idea is to feed an LLM the fund prospectus, marketing materials, and portfolio data, and ask it to produce the EMT or EET values directly. The problem is that regulatory disclosures have legal weight — the producer is formally asserting, with regulatory consequences for error, that the values are correct. An LLM can produce plausible-looking EMT values that are subtly wrong (a risk indicator shifted by one level, a SFDR classification misread, a cost figure that looks right but fails to account for a hedging arrangement), and the plausibility of the output makes the error harder to catch than a crude bug would be. Regulatory content must be generated by deterministic logic against trusted source data, not by a probabilistic model, and the prohibition should be treated as absolute until the accountability chain for AI-generated regulatory content is much clearer than it is today.

Automatic correction of validation failures. Another tempting idea is to feed a validation failure back to an LLM and ask it to fix the FundsXML document so that validation passes. This works technically — the model can produce a modified document that satisfies the schema — but the modifications may silently change the meaning of the data. A missing element gets filled in with a plausible default value that happens to be wrong; an out-of-range number gets clipped to the nearest valid value, losing the signal that the original number was a bug in the upstream system; a type

error gets fixed by coercion that hides a deeper data-quality issue. The validation failure was a symptom of a real problem, and masking it with an AI fix is how producers emit bad data that passes validation. Validation failures must be investigated and fixed by humans who understand what the data should be, not patched away automatically.

Anomaly detection without explainability. AI-based anomaly detection — training a model on historical FundsXML deliveries and flagging the ones that look unusual — has legitimate uses, and several vendors are building products in this space. The caveat is that any anomaly detector deployed in an operational role must be able to *explain why* it flagged a particular delivery. An alert that says "delivery for 2026-03-31 looks anomalous, confidence 0.87" is worse than useless: it creates work for the on-call engineer without giving them any way to investigate. An alert that says "delivery for 2026-03-31 has a TNAV change of 12% compared to 2026-02-28, which is three standard deviations outside the historical distribution for this fund" is useful, because it tells the engineer exactly what to check. Explainability is not optional; it is the difference between a useful tool and noise.

14.4.3 The Long View on Automation

Beyond the specific applications of AI, a broader question worth asking is what the *overall* trajectory of automation in FundsXML work will look like over the next five to ten years. The answer most honest to the evidence is that automation will eliminate a lot of the tedious work (mapping, rule writing, change-impact analysis, documentation generation) while leaving the judgement-heavy work (scope decisions, architectural trade-offs, stakeholder negotiation, root-cause analysis of real incidents) firmly in human hands. The working life of a FundsXML engineer in 2030 will probably involve less typing than the working life in 2026 but more careful thinking. The skill set shifts from *knowing how to write the mapping* to *knowing how to review a mapping well*, and the latter is a more valuable skill than the former.

This is an optimistic reading, and it may be wrong. The pessimistic reading is that automation will hollow out the middle layer of the engineering career, leaving a small number of senior engineers who guide the automation and a large number of junior engineers who operate it without understanding it, with no clear path from junior to senior because the experience that used to build judgement (actually writing the mapping, actually debugging the validation failure) has been automated away. This is a real risk, and the community should take it seriously. The countermeasure is deliberate: use the automation to eliminate the tedium, but preserve the learning opportunities by having junior engineers review the automation's output critically rather than rubber-stamping it.

Neither the optimistic nor the pessimistic reading is certain, and the actual outcome will depend on choices that individual teams make over the coming years. The book you are holding is a small contribution to the optimistic reading: if more practitioners understand the schema and the implementation process in depth, the judgement-heavy work has a larger pool of people capable of doing it well, and the automation becomes a lever rather than a replacement.

14.5 How to Contribute

14.5.1 The Community Is Open

A recurring frustration among newcomers to any technical standard is the feeling that the standard is *done* — that it was decided by experts in a closed room and handed to the rest of the world to implement. For some standards, this is accurate. For FundsXML, it is not. The community that shapes the schema is genuinely open to contributions from practitioners at any level, and the barriers to participation are lower than most newcomers expect.

This section lists, in order of increasing commitment, the concrete ways in which a reader of this book can become a contributor to the FundsXML community. None of them require formal credentials, none require being employed by a member organisation, and all of them are valued by the community that currently does the work.

14.5.2 Filing an Issue

The lowest-effort contribution is also, in many ways, the most valuable: **file an issue when you find a problem**. The FundsXML association maintains a public issue tracker on GitHub where any reader can report a problem with the schema, the documentation, the sample files, or the tooling. A good issue includes a clear description of the problem, a minimal reproduction if applicable, and a suggestion of what the correct behaviour should be. The issue will be triaged by a maintainer within a few days, discussed in the relevant working group, and either accepted for a future release, accepted with modifications, or declined with a written explanation.

The value of issues is that they surface problems the maintainers cannot see from inside the project. A producer team in Lisbon that runs into an ambiguous element in the EMT block has information the Vienna-based maintainers do not have, and filing an issue is the mechanism by which that information reaches the people who can act on it. A consumer team in Stockholm that finds a documentation gap has information that only a careful reader can produce. Every release cycle of FundsXML incorporates fixes and clarifications that originated as issues from practitioners, and the practitioners who filed them became, in doing so, contributors to the standard.

A brief etiquette note: an issue that says "the schema is wrong" without further detail is not useful, and an issue that is really a question about how to use the schema correctly should be asked on the mailing list rather than filed as an issue. But an issue that says "in version 4.2.8, element X allows enumeration value Y but the description in the associated documentation says Y is only valid in combination with Z, which is not reflected in the schema constraints" is exactly the kind of precise, actionable report that maintainers love to receive.

14.5.3 Joining the Mailing List and Discussion Channels

A step up in commitment is joining the FundsXML community mailing list and, where available, the chat channels (Slack or Discord, depending on the working group). The mailing list is where announcements happen, where discussions about upcoming changes are hashed out, and where newcomers can ask questions without filing formal issues. Reading the list for a few months before posting is the standard advice for any new member of any technical community, and FundsXML is no exception; the discussions have a context that builds up over time, and lurking long enough to understand the context makes later contributions more useful.

Once you are comfortable with the discussion style, participating in the mailing list is a genuine form of contribution. A single well-phrased question can help dozens of other lurkers who had the same question but had not asked it. A thoughtful reply to someone else's question is often more valuable than the original question, because it captures the implicit knowledge of an experienced practitioner in a form that the less experienced can learn from. The FundsXML mailing list has a handful of people who consistently do this — who answer questions patiently, explain the reasoning behind schema decisions, and point newcomers at the right parts of the documentation — and those people are among the most respected members of the community even though their formal role in the association may be modest.

14.5.4 Joining a Working Group

The next step up is joining one of the FundsXML association's working groups. Working groups are standing committees of practitioners who meet regularly (usually monthly, usually by video conference) to discuss specific topics — schema evolution, regulatory module coordination with FinDatEx, tooling, documentation, and so on. Membership is open to employees of association member organisations and, on a case-by-case basis, to individual practitioners who can contribute relevant expertise. The time commitment is modest — a few hours per month — and the learning is considerable, because working-group discussions expose participants to the reasoning behind schema decisions in a way that reading the final release notes never can.

Working groups are also where the actual work of schema evolution happens. A proposal for a new element, a change to an enumeration, a deprecation of an old construct — all of these start as conversations in a working group, are refined through several iterations, and eventually become part of a release. A practitioner who has an idea for improving the schema can propose it directly in the relevant working group and see it through to a release, and this path from idea to released schema is genuinely accessible to anyone willing to engage.

14.5.5 Contributing to Open-Source Tooling

Beyond the schema itself, the FundsXML ecosystem includes a growing set of open-source tools, the most prominent of which — FreeXmlToolkit — was covered in Chapter 11. These tools accept contributions through their respective GitHub repositories, and contributions come in every size: bug reports, bug fixes, documentation improvements, translations, new features, performance improvements, test-case additions. A practitioner who uses FreeXmlToolkit daily is in a better position to notice usability issues and to fix small bugs than the tool's author is, and a steady stream of small contributions from daily users is what turns a one-person side project into a mature community tool.

The skills needed for tooling contributions depend on the tool. FreeXmlToolkit is written in Java with JavaFX, so Java skills are the primary technical prerequisite. Other tools in the ecosystem are written in Python, JavaScript, or C#, and each has its own contribution process. A reader who wants to contribute but is unsure where to start can post on the tool's issue tracker or mailing list asking for suggestions — maintainers typically have a short list of "good first issue" tasks that they save for newcomers precisely because newcomer contributions are valuable and deserve to be welcomed.

14.5.6 Teaching and Writing

A different kind of contribution, often overlooked but genuinely valuable, is **teaching and writing about FundsXML**. The community grows when new practitioners learn the schema and become productive, and learning is accelerated when good teaching materials exist. A blog post explaining a specific corner of the schema, a conference talk describing an implementation project, a tutorial video showing how to use a particular tool, a translation of the documentation into a language not currently covered — all of these are contributions, and all of them reach audiences that the formal documentation does not.

The book you are reading is an instance of this kind of contribution: it exists because a group of practitioners decided that the existing learning materials were insufficient for newcomers and took the time to write a book-length introduction. It is not the only way to contribute through writing, and a reader need not aim at a full book to make a useful written contribution. A single well-crafted blog post explaining a concept that the official documentation glosses over can help hundreds of newcomers in a way that no committee meeting ever will.

14.5.7 Hypothetical Europa Contributions

To make this concrete in the book's running example: the Europa Asset Management implementation project described in Chapter 13 produced a number of contributions that the team *could* have returned to the community if they had chosen to. They developed a detailed mapping table with 450 rows, which — with the source-column names redacted — could have been a valuable starting template for other asset managers undertaking similar projects. They wrote a Schematron rule set covering several dozen business rules, which could have been published as a reference rule set. They discovered and filed four schema ambiguities during the mapping phase, which they resolved internally but did not file as issues. They built a REST facade in early 2027 that could have been open-sourced as a reference implementation of the "batch-XML plus API-JSON" pattern described in §14.3.

None of these hypothetical contributions happened, because the Europa project was focused on shipping and the team did not have time to package its work for external release. This is the usual reason contributions do not happen: the team that has the knowledge also has the pressure of a live project, and the overhead of turning internal work into external contribution feels too high. The counter-argument is that contributions, done well, are a small investment that compounds: a reference Schematron rule set published in 2026 saves future teams weeks of work in 2027, 2028, and beyond, and the publishing team gains reputation in the community that translates into easier hiring, easier peer support, and easier access to the people shaping the standard. A reader of this book who is running a FundsXML project of their own should take a few minutes to consider which of their project's outputs might be worth contributing back, and schedule the packaging work *during* the project rather than after the team disbands.

14.6 Key Takeaways

- The FundsXML standard is in continuous evolution, with two to three minor releases per year coordinated by the FundsXML association. A producer or consumer that freezes against a single release will eventually fall behind the rest of the ecosystem; staying current is a recurring maintenance task, not a one-time event.

- The 2026 roadmap discussions cluster around four themes: **digital and tokenised assets, ESG and sustainability extensions, T+1 settlement, and cross-border distribution under the revised UCITS framework**. None is guaranteed to land, but all are worth watching.
- The XML-versus-JSON debate is best answered as "both, for different purposes": FundsXML's core value is the data model, the XML serialisation suits batch regulatory delivery, and JSON projections via REST or GraphQL facades serve API-driven use cases. The two can coexist, and most mature producers will eventually run both.
- The **European Single Access Point (ESAP)**, due to become operational in 2027, will shift FundsXML from an industry-internal format toward a public-good format. This raises the stakes of data quality, schema readability, and community governance.
- **AI and automation** are already helping with mapping, schema-migration analysis, and Schematron rule suggestion — always as first-draft generators with a human in the loop. They should **not** be trusted with direct generation of regulatory content, automatic patching of validation failures, or anomaly detection without explainability. The working life of a FundsXML engineer in 2030 will involve less typing and more judgement.
- **Contributing to the community is open to any reader of this book**, regardless of employer or credentials. The lowest-effort contribution — filing a well-phrased issue when you find a problem — is also one of the most valuable. Higher-commitment contributions include joining the mailing list, participating in working groups, contributing to open-source tools like FreeXmlToolkit, and writing or teaching about the standard.
- The Europa Growth Fund project described in Chapter 13 *could* have produced several valuable community contributions (a reference mapping template, a published Schematron rule set, filed schema-ambiguity issues, an open-source REST facade). It did not, because the team was focused on shipping. The lesson for the reader running a similar project: schedule the packaging work *during* the project, not after, and contribute back.

This chapter closes the narrative portion of the book. What follows — Appendices A through F — is reference material: a glossary, an XML and XPath cheat sheet, a structured schema overview, complete Europa Growth Fund sample files, a directory of resources and links, and a final reference section. The appendices are meant to be consulted rather than read through; you will find yourself returning to them as your own FundsXML projects raise specific questions that the narrative chapters have primed you to ask.

Thank you for reading. The standard belongs to everyone who uses it, and now that includes you.

Glossary

Technical terms in German and English

This glossary is a reference, not a second introductory chapter. Each entry gives a short definition — usually one to three sentences — together with cross-references to other glossary entries (marked with →) and, where appropriate, to the chapter in the main text where the term is treated in depth.

Conventions. Entries are sorted alphabetically by their English term, which is shown in **bold**. The German equivalent, where one exists in common use, follows in italic parentheses immediately after the English lemma: **Fund** (*Fonds*). Where the English term is itself a loan-word used unchanged in German, the italic is omitted. Cross-references use the arrow symbol →, which points at another entry in this glossary or, less frequently, at a numbered chapter or section of the book. The acronyms list in §A.2 collects every abbreviation used in the book in one place for quick lookup.

A final note on the language choice. The main text of this book is written in British English, but the FundsXML community is historically rooted in German-speaking Central Europe, and many of the working documents that practitioners encounter — depositary agreements, fund prospectuses, BaFin circulars, Austrian FMA guidance — are in German. The bilingual glossary is meant to bridge the two vocabularies so that a practitioner can move between an English specification and a German working document without losing the thread.

A.1 Alphabetical Entries

A

Accounting NAV (*Buchhalterischer NAV*) — The net asset value calculated by the fund's accounting system before any adjustments for swing pricing, rounding conventions, or other publication-time modifications. Contrast with the *published NAV*, which is what reaches investors. → NAV, → Swing Pricing.

Accumulating Share Class (*Thesaurierende Anteilklasse*) — A share class in which income (dividends, interest, realised gains) is reinvested into the fund rather than distributed to the investor. The investor's return is reflected in an increasing NAV per share rather than in periodic cash payments. Contrast with → Distributing Share Class.

Administrator — See → Fund Administrator.

AIF — Alternative Investment Fund. Any collective investment undertaking in the EU that does not qualify as a UCITS fund — typically hedge funds, private equity funds, real estate funds, infrastructure funds, and most institutional-only funds. Regulated under → AIFMD.

AIFMD — Alternative Investment Fund Managers Directive. The EU directive (2011/61/EU) governing the management and marketing of → AIF funds in the European Union. Its successor, AIFMD II, was adopted in 2024. → Chapter 1 for context.

Asset Manager (*Kapitalverwaltungsgesellschaft, KVG*) — The firm that manages a fund's investment portfolio according to its prospectus and investment guidelines. In the FundsXML data model, the asset manager is typically the → [Data Supplier](#) for fund-level data. → Chapter 1.

Asset Master Data — The static reference data about investable instruments (equities, bonds, derivatives) that a portfolio may contain: ISIN, name, issuer, country of risk, classification codes. In FundsXML, this data lives in the `AssetMasterData` block rather than being duplicated inside each position. → Chapter 6.

Audit Log (*Audit-Protokoll*) — A chronological, tamper-evident record of every delivery emitted or received by a FundsXML pipeline, together with its status, validation result, and any operator actions. Required for most regulatory frameworks and practically necessary for operational troubleshooting. → Chapter 13.

B

Base Currency (*Basiswährung*) — The currency in which a fund's accounts are kept and in which its primary NAV is expressed. Individual share classes may be denominated in other currencies, in which case the fund-level TNAV is a base-currency figure and the share-class TNAVs are in their own currencies. → [Share Class](#), → [TNAV](#).

Batch Delivery (*Stapellieferung*) — A mode of data exchange in which producers emit complete data sets at scheduled intervals (daily, monthly, quarterly) rather than streaming individual events. The dominant pattern in fund data exchange and the pattern for which the FundsXML schema was designed. Contrast with → [Event-Driven Architecture](#).

Business Rule (*Geschäftsregel*) — A correctness requirement that goes beyond structural validation and expresses domain-specific logic (e.g., "a DELETE operation must reference the document being retracted"). Business rules are typically enforced with → [Schematron](#) rather than with XSD. → Chapter 10.

C

Cash Equivalent (*Zahlungsmitteläquivalent*) — Short-term, highly liquid instruments (money-market funds, overnight deposits, T-bills with less than three months to maturity) that are treated as cash in portfolio reporting. Carried in FundsXML as a position with a `PositionType` indicating the cash-equivalent classification.

Consumer (*Datenempfänger*) — A system or organisation that receives FundsXML files and uses them for downstream processing: a distributor generating KIDs, a data warehouse loading historical positions, a regulator ingesting disclosures for oversight. Contrast with → [Producer](#). → Chapter 1.

ControlData — The envelope block at the top of every FundsXML document that identifies the delivery: unique document ID, content date, data supplier, data operation, language, version. The dispatcher's first anchor when routing an incoming delivery. → Chapter 4.

Corporate Action (*Kapitalmaßnahme*) — An event that changes the terms of a security held in a portfolio: a stock split, a dividend, a merger, a spin-off, a name change. Corporate actions are recorded in FundsXML as transaction-like events that adjust positions between valuation dates. → Chapter 7.

Cross-Border Distribution (*Grenzüberschreitender Vertrieb*) — The authorised offering of a fund's shares to investors in an EU member state other than the fund's country of domicile. Governed by the UCITS or AIFMD passporting regime. Each distribution country may impose its own information requirements, which FundsXML carries

in `CountrySpecificData` . → Chapter 9.

Custodian (*Depotbank, Verwahrstelle*) — The bank that holds a fund's assets in safe-keeping and verifies ownership on behalf of investors. Under UCITS and AIFMD, funds must appoint a single depositary that is legally responsible for asset safe-keeping, cash monitoring, and oversight of the fund manager. Often used interchangeably with *depository* in English, though strictly speaking the depositary has broader duties. → Chapter 1.

CustomAttributes — The FundsXML extension mechanism for carrying producer- or project-specific data that the standard schema does not model natively. Each custom attribute has a name, a type indicator (T/N/D/B for text/number/date/boolean), and a typed value. To be used sparingly; the first question should always be whether a standard element already exists. → Chapter 9.

Cutoff Time (*Order-Annahmeschluss*) — The deadline each day by which investor orders (subscriptions or redemptions) must be received to be executed at that day's NAV. Orders after cutoff are held for the next NAV calculation. Relevant to FundsXML because it affects which transactions appear in which delivery's valuation period.

Cutover — The transition moment at which a new pipeline takes over from a legacy one. Contrast with → Parallel Run, which keeps both pipelines running for a period. → Chapter 13.

D

Data Gap (*Datenlücke*) — A FundsXML element that a delivery needs to populate but for which no existing source system holds the value. Data gaps are typically surfaced during the mapping phase of an implementation project and must be resolved before production. → Chapter 13.

Data Operation — One of three enumerated values — `INITIAL` , `AMEND` , `DELETE` — in `ControlData/DataOperation` that tells the consumer what the delivery is meant to do: establish a new record, update an existing record, or retract one. Confusing `AMEND` with a silent overwrite is a classic pipeline bug. → Chapter 4.

Data Quality (*Datenqualität*) — The property that data is accurate, complete, consistent, and timely. Data-quality issues, unlike data gaps, involve fields that do exist but hold incorrect or inconsistent values — for example, LEIs that are missing check digits or currency codes that do not match ISO 4217. → Chapter 13.

Data Supplier (*Datenlieferant*) — The organisation that generated a particular FundsXML delivery, identified in `ControlData/DataSupplier` by system country, short code, legal name, and type. Not necessarily the asset manager — it might be a fund administrator or a reporting agent acting on the manager's behalf. → Chapter 4.

Depository — See → Custodian.

Dispatcher (*Verteiler*) — A consumer-side component that receives incoming FundsXML files and routes them to the correct downstream systems based on their content — distributor KID generator, trading desk, warehouse loader, and so on. → Chapter 12.

Distributing Share Class (*Ausschüttende Anteilklasse*) — A share class that pays out its income to investors periodically (usually annually or semi-annually) rather than reinvesting it. Contrast with → Accumulating Share Class.

Distribution Country (*Vertriebsland*) — A country in which a fund's shares are authorised for offering to investors. Each distribution country may impose local information requirements beyond those of the fund's domicile. → Chapter 9.

DocumentGenerated — The timestamp, in `ControlData/DocumentGenerated`, at which the FundsXML file was created by the producer. Contrast with `→ ContentDate`, which is the valuation date the delivery describes.

Domicile (*Sitzstaat*) — The country in which a fund is legally established and whose regulator primarily supervises it. Luxembourg, Ireland, France, Germany, and Austria account for most UCITS domiciles in the EU.

E

EET — European ESG Template. The FinDatEx template that carries the SFDR-mandated sustainability disclosures for a fund: PAI indicators, Taxonomy alignment, sustainable investment share, exclusions policy, and related fields. `→ Chapter 8`.

EFT — European Feeder Fund Template. The FinDatEx template for communicating feeder-fund-specific information between a feeder fund and its master. `→ Chapter 8`.

EMT — European MiFID Template. The FinDatEx template that carries the MiFID II target market, cost, and risk-classification information that distributors need for investor-suitability assessments and for PRIIPs KIDs. The most widely used of the FinDatEx templates. `→ Chapter 8`.

Emission Window (*Lieferfenster*) — The operational time-slot within which a pipeline is expected to emit its daily or monthly delivery. Failures within the emission window typically trigger an on-call page; failures outside the window can wait for business hours. `→ Chapter 13`.

EPT — European PRIIPs Template. The FinDatEx template that carries the inputs a distributor needs to generate a PRIIPs Key Information Document (KID). `→ Chapter 8`.

ESAP — European Single Access Point. A centralised EU platform, established by Regulation 2023/2859 and scheduled to become operational in 2027, that aggregates regulatory disclosures from all EU-listed entities and makes them publicly searchable. Funds are within its scope. `→ Chapter 14`.

ESG — Environmental, Social, and Governance. The umbrella term for non-financial investment criteria covering sustainability, social impact, and corporate governance. Reflected in FundsXML primarily through the EET module and the SFDR disclosures. `→ Chapter 8`, `→ Chapter 14`.

ESMA — European Securities and Markets Authority. The EU authority that issues technical standards and guidance for securities markets, including the regulatory technical standards (RTS) underlying SFDR, MiFID II, PRIIPs, and UCITS. `→ Chapter 1`.

Event-Driven Architecture — A system design in which state changes are published as real-time events on a message bus (Kafka, Kinesis, RabbitMQ) rather than aggregated into batch files. FundsXML is designed for batch delivery and does not currently have a first-class event mode. `→ Chapter 14`.

Exchange-Traded Fund (ETF) (*Börsengehandelter Fonds*) — A fund whose shares are listed on a stock exchange and traded intraday like equities, in contrast to traditional mutual funds that are bought and sold at a single end-of-day NAV. Most European ETFs are structured as UCITS funds.

F

Fair Value (*Beizulegender Zeitwert*) — The value at which an asset could be exchanged between knowledgeable, willing parties in an arm's-length transaction. For actively traded instruments, fair value is the market price; for illiquid instruments, it is estimated using valuation models. The basis on which NAV is calculated. → NAV.

Feeder Fund (*Feeder-Fonds*) — A fund that invests substantially all of its assets in a single other fund (the → Master Fund), typically to give investors in one jurisdiction access to a master fund domiciled elsewhere. Reflected in FundsXML via the EFT module for feeder-specific information.

FIGI — Financial Instrument Global Identifier. A 12-character alphanumeric identifier for financial instruments, issued by Bloomberg and standardised as an open identifier. Less common in European fund data than ISIN but growing in use. → ISIN.

FinDatEx — European Working Group on Investment Data Exchange. The industry body that publishes the five European data templates (EMT, EPT, EET, EFT, TPT) embedded in the FundsXML [RegulatoryReportings](#) block. Coordinates its release cycle with the FundsXML association. → Chapter 8, → Chapter 14.

Fixture (*Testfixture*) — A pre-prepared test input (a FundsXML file, a database snapshot, a configuration) used to exercise a specific test case reproducibly. The fixture repository of a mature pipeline is the institutional memory of "things that have gone wrong". → Chapter 13.

FreeXmlToolkit — An open-source desktop application by Karl Kauc for working with XML, XSD, XSLT, Schematron, and FundsXML files. Covered in detail in → Chapter 11.

Fund (*Fonds*) — A collective investment undertaking that pools capital from investors and invests it according to a stated strategy. In FundsXML, represented by the `Fund` element with its static and dynamic sub-blocks. → Chapter 5.

Fund Administrator (*Fondsbuchhalter, Verwaltungsstelle*) — The firm that provides back-office services to a fund: NAV calculation, fund accounting, investor register maintenance, regulatory reporting. In practice, the fund administrator is often the producer of FundsXML deliveries on behalf of the asset manager. → Chapter 1.

FundDynamicData — The FundsXML block inside each `Fund` element that carries values changing at each valuation: TNAV, share counts, prices, flows, and earnings. Contrast with → [FundStaticData](#). → Chapter 5.

FundStaticData — The FundsXML block inside each `Fund` element that carries values which change rarely: fund name, domicile, launch date, investment manager, custodian, prospectus references. → Chapter 5.

FundsXML — The XML-based open standard for fund data exchange maintained by the FundsXML association. The subject of this book.

FundsXML Association (*FundsXML-Verein*) — The non-profit body, registered in Austria, that stewards the FundsXML standard: governance, working groups, release coordination, consultation process, documentation. → Chapter 14.

G

Generator — (1) A producer-side component that assembles source data into a FundsXML document. (2) The FreeXmlToolkit feature that creates synthetic FundsXML documents from a schema, useful for testing. → Chapter 11.

GLEIF — Global Legal Entity Identifier Foundation. The non-profit body that oversees the global LEI system, including the issuance and verification of LEIs. → LEI.

Golden Fixture — A reference FundsXML file whose content is considered correct and against which newer pipeline outputs are compared byte-for-byte or field-by-field in regression testing. → Chapter 13.

H

Hedged Share Class (*Währungsgesicherte Anteilklasse*) — A share class whose currency exposure is hedged back to the investor's reference currency, so that the investor experiences the fund's underlying returns without the additional currency-translation risk. → Share Class.

I

Identifier (*Kennung*) — A code that uniquely identifies a fund, share class, instrument, or legal entity. FundsXML uses LEI for legal entities, ISIN for share classes and instruments, and a small number of additional identifier types (CUSIP, SEDOL, WKN, FIGI, CFI) as alternatives or supplements.

Idempotency (*Idempotenz*) — The property that applying the same operation multiple times produces the same result as applying it once. A consumer pipeline that handles idempotency correctly will not duplicate rows when a delivery is re-sent. → Chapter 12.

INITIAL — One of three values of → Data Operation, indicating that the delivery introduces new data rather than modifying or retracting an earlier delivery. The default for a normal periodic delivery.

Integration Test (*Integrationstest*) — A test that exercises multiple components of a pipeline together, typically end-to-end from source database to validated output file. Contrast with a → Unit Test, which exercises a single component in isolation. → Chapter 13.

ISIN — International Securities Identification Number. A 12-character alphanumeric code (ISO 6166) that uniquely identifies a security globally. FundsXML uses ISINs for share classes and portfolio instruments. → Chapter 5.

K

KID — Key Information Document. The standardised three-page investor information document required under the PRIIPs regulation for retail investment products, including UCITS funds. KID content is generated from → EPT template data. → Chapter 8.

Knowledge Transfer (*Wissenstransfer*) — The deliberate process of moving institutional knowledge from a project team to the permanent operations team before the project team disbands. A critical but often neglected phase of an implementation project. → Chapter 13.

L

LEI — Legal Entity Identifier. A 20-character alphanumeric code (ISO 17442) that uniquely identifies a legal entity in financial markets. Mandatory for fund management companies, funds, and counterparties to most regulated transactions. → Chapter 4, → Chapter 5.

Life-Cycle (*Lebenszyklus*) — The sequence of states a fund moves through from launch to liquidation: in-registration, launched, active, in-liquidation, liquidated. FundsXML does not carry a formal life-cycle enumeration, but some status information is captured in the fund static data.

M

Management Company — See → Asset Manager.

Mapping (*Abbildung*) — The systematic connection between each FundsXML element that a pipeline must populate and the source-system field from which its value is drawn. Typically captured in a mapping table with columns for path, source system, source column, transformation rule, and default. → Chapter 13.

Master Fund — The fund in which one or more → Feeder Funds invest substantially all of their assets. A master-feeder structure is typically used for cross-border distribution efficiency.

MiFID II — Markets in Financial Instruments Directive II. The EU directive (2014/65/EU) that governs investment services and regulated markets, including the target-market and cost-disclosure requirements that the → EMT template carries. → Chapter 8.

N

Namespace (*Namensraum*) — An XML mechanism for scoping element and attribute names to a specific vocabulary, identified by a URI. FundsXML 4.2 is defined without a target namespace, which is unusual for a modern XML standard but simplifies tooling. → Chapter 4.

NAV — Net Asset Value (*Nettoinventarwert*). The total value of a fund's assets minus its liabilities, usually expressed both as a total figure (→ TNAV) and as a per-share figure (NAV per share). The central number in fund accounting and the anchor of every FundsXML dynamic-data delivery. → Chapter 5.

Non-Functional Requirement (*Nicht-funktionale Anforderung*) — A requirement about how a system operates rather than what it does: performance, availability, security, monitoring, audit retention. → Chapter 13.

O

OFFICIAL — One of three values of `TotalAssetNature` (together with `ESTIMATED` and `TECHNICAL`), indicating that the NAV in the delivery is the final, official figure that investors will transact against. → Chapter 5.

Operations Team (*Betriebsteam*) — The permanent team that runs the production pipeline after the project team has disbanded. Its role, capabilities, and relationship to the builders are central to the project-to-operations transition. → Chapter 13.

P

PAI — Principal Adverse Impacts. A defined set of sustainability indicators, mandated by SFDR, that asset managers must disclose for investment products. PAI values are carried in the → EET template. → Chapter 8.

Parallel Run (*Parallelbetrieb*) — A go-live strategy in which both the legacy pipeline and the new pipeline emit deliveries for the same period, allowing daily comparison and a safety net if the new pipeline fails. The professional standard for FundsXML go-lives. → Chapter 13.

Paying Agent (*Zahlstelle*) — A local agent in a distribution country that handles subscription payments, redemption payments, and distributions on behalf of investors in that country. A legal requirement for cross-border UCITS distribution in some jurisdictions.

Pipeline — The end-to-end sequence of components that move data from source to destination: in the producer case, from internal databases through aggregation, generation, validation, and emission; in the consumer case, from ingestion through parsing, validation, and loading. → Chapter 12.

Portfolio (*Portfolio, Vermögensaufstellung*) — The full list of positions held by a fund at a given valuation date. In FundsXML, carried in the **Portfolios** block as one or more **Portfolio** elements, each containing multiple **Position** entries. → Chapter 6.

Position (*Position*) — A specific holding of a specific instrument within a portfolio: the quantity, the unit price, the market value, and any associated classifications. → Chapter 6.

PRIIPs — Packaged Retail and Insurance-based Investment Products. The EU regulation (1286/2014) that requires a standardised → KID for retail investment products. → EPT template data is the input. → Chapter 8.

Producer (*Datenlieferant*) — A system or organisation that generates FundsXML deliveries: asset manager, fund administrator, reporting agent. Contrast with → Consumer. → Chapter 1.

Prospectus (*Verkaufsprospekt*) — The official legal document describing a fund's investment strategy, risks, costs, and structure, required by UCITS and AIFMD for investor disclosure. Referenced from FundsXML by URL or document identifier rather than embedded.

Prototype (*Prototyp*) — A throwaway implementation built during Phase 3 of an implementation project to discover the problems that only become visible when code meets real data. Deliberately separate from the production build. → Chapter 13.

R

Redemption (*Rücknahme*) — An investor's sale of fund shares back to the fund at the current NAV, reducing the fund's TNAV and share count. Recorded in FundsXML in the dynamic-data flows block. → Chapter 7.

RegulatoryReportings — The FundsXML block that carries the five FinDatEx regulatory templates (EMT, EPT, EET, EFT, TPT) for a fund. Embedded rather than referenced, so the full regulatory content travels with the delivery. → Chapter 8.

RelatedDocumentIDs — A FundsXML element that back-references a previous delivery whose content the current delivery modifies or replaces. Required for AMEND and DELETE operations so that the consumer can locate the original. → Chapter 4, → Chapter 10.

Runbook (*Betriebshandbuch*) — The operational manual for a pipeline: how to start and stop it, how to investigate failures, how to re-emit deliveries, how to roll back in an emergency. Written during the testing phase, delivered to operations at go-live. → Chapter 13.

S

Schema (*Schema*) — In the XML context, a formal description of the permitted structure and content of a class of documents. The FundsXML schema is defined in XSD and is the authoritative specification of what a valid FundsXML document looks like. → XSD.

Schematron — A rule-based validation language that expresses business rules as XPath assertions, complementing the structural validation that XSD provides. The second stage of the two-stage validation pattern. → Chapter 10.

SFDR — Sustainable Finance Disclosure Regulation. The EU regulation (2019/2088) that requires asset managers to disclose how they integrate sustainability risks and adverse impacts into their investment processes. The regulatory driver behind the → EET template. → Chapter 8.

Share Class (*Anteilklasse*) — A sub-category of a fund offering different terms to different investor groups: different fees, different currencies, different distribution policies, different minimum investments. Each share class has its own ISIN and its own NAV per share. → Chapter 5.

Signature (*Signatur*) — A cryptographic proof that a document has not been altered since it was signed. FundsXML supports XML Digital Signatures (XMLDSig) at the document root, so that consumers can verify provenance and integrity. → Chapter 9.

Source System (*Quellsystem*) — The internal system (portfolio management, client reference database, regulatory data mart, spreadsheet) from which a producer pipeline reads its input data. The mapping table is the contract between the source systems and the pipeline. → Chapter 13.

Subscription (*Zeichnung*) — An investor's purchase of newly issued fund shares from the fund at the current NAV, increasing the fund's TNAV and share count. Contrast with → Redemption. → Chapter 7.

Sub-Fund (*Teilfonds*) — A fund that is part of an → Umbrella Fund structure, with its own investment strategy, portfolio, and NAV but sharing the umbrella's legal personality, prospectus, and management company.

Swing Pricing (*Swing Pricing*) — A technique in which the fund's NAV is adjusted on days of large net subscriptions or redemptions to protect existing shareholders from the transaction costs caused by other investors' flows. Applied on top of the accounting NAV to produce the published NAV.

T

Target Market (*Zielmarkt*) — The category of investors for which a fund is considered suitable, as defined by MiFID II and carried in the → EMT template: investor type, knowledge and experience, financial situation, risk tolerance, investment objective, distribution strategy. → Chapter 8.

T+1 / T+2 (*T+1 / T+2*) — Settlement conventions indicating that a trade settles one (T+1) or two (T+2) business days after trade date. European equities moved to T+2 years ago; the US moved to T+1 in 2024; Europe is scheduled to follow in 2027. → Chapter 14.

TNAV — Total Net Asset Value. The fund-level aggregate of net asset value across all share classes, expressed in the fund's base currency. Carried in FundsXML in `FundDynamicData/TotalAssetValues/TotalAssetValue`. → Chapter 5.

TPT — Tripartite Template. The FinDatEx template used for institutional reporting of portfolio composition between an asset manager, its insurance client, and the client's regulator. Named for its three-party audience. → Chapter 8.

Transaction (*Transaktion*) — A trade or capital event that changes the composition or size of a fund's portfolio: a buy, a sell, a subscription, a redemption, a corporate action. Recorded in FundsXML in the `Transactions` block. → Chapter 7.

Transfer Agent (*Transferstelle*) — The firm that maintains the register of a fund's investors and processes subscriptions, redemptions, and transfers between investors. Typically part of the fund administrator's service offering but legally a distinct role.

Two-Stage Validation — The validation pattern in which a FundsXML document is first checked against the XSD schema for structural correctness and then against a Schematron rule set for business-rule correctness. Only documents that pass both stages are considered valid. → Chapter 10.

U

UAT — User Acceptance Testing. The phase of an implementation project in which stakeholders review the pipeline's output and formally sign off that it meets their expectations. A gate before go-live. → Chapter 13.

UCITS — Undertakings for Collective Investment in Transferable Securities. The EU directive framework (originally 1985, most recently consolidated as 2009/65/EC) governing retail investment funds that can be distributed cross-border within the EU on the basis of a single authorisation. The dominant European fund structure. → Chapter 1.

Umbrella Fund (*Umbrella-Fonds, Dachfonds*) — A legal fund structure that contains multiple → Sub-Funds under a single legal personality and a single prospectus. Common in Luxembourg and Ireland for operational and marketing efficiency.

Unit Test (*Unit-Test*) — A test that exercises a single small component (a single transformation rule, a single function) in isolation, giving fast and precise feedback about whether that component is correct. Contrast with → Integration Test. → Chapter 13.

V

Validation (*Validierung*) — The process of checking that a FundsXML document is both structurally correct (against XSD) and semantically correct (against Schematron business rules). Distinct from *verification* of the data itself, which is an upstream concern. → Chapter 10.

Valuation Point (*Bewertungszeitpunkt*) — The moment in time, usually a specific time of day, at which a fund's NAV is calculated. Orders received before the cutoff time are executed at the NAV from the next valuation point. → Cutoff Time.

W

WKN — Wertpapierkennnummer. The six-character German national security identifier, historically preceding ISIN and still used alongside ISIN in German-speaking markets. A secondary identifier in FundsXML.

Working Group (*Arbeitsgruppe*) — A standing committee of practitioners within the FundsXML association (or FinDatEx) that meets regularly to discuss a specific topic area — schema evolution, regulatory coordination, tooling — and to draft proposals for the next release. → Chapter 14.

X

XML — Extensible Markup Language. The W3C standard for structured text documents on which FundsXML is built. → Chapter 2.

XML Digital Signature — See → Signature.

XPath — A W3C standard query language for selecting nodes in an XML document. The expression language used by Schematron and by most XML processing libraries. → Chapter 2, → Chapter 10.

XSD — XML Schema Definition. The W3C standard schema language in which the FundsXML schema is written. Describes element and attribute structure, data types, cardinalities, and enumerations. → Chapter 2, → Chapter 10.

XSLT — Extensible Stylesheet Language Transformations. The W3C standard language for transforming XML documents into other XML documents (or HTML, or text). Useful for producing human-readable reports from FundsXML files or for converting between schema versions. → Chapter 12 §12.3.7 for worked examples, → Chapter 11 for the FreeXmlToolkit XSLT Developer tab.

A.2 Acronyms

The following acronyms recur throughout the book. Where a term is treated in detail elsewhere in this glossary or in the main text, the relevant reference is given.

- **AIF** — Alternative Investment Fund → A.1
- **AIFMD** — Alternative Investment Fund Managers Directive → A.1, Chapter 1
- **BaFin** — Bundesanstalt für Finanzdienstleistungsaufsicht (German financial supervisory authority)
- **CFI** — Classification of Financial Instruments (ISO 10962)
- **CRD** — Client Reference Database (a type of internal source system, not a standard acronym)
- **CSV** — Comma-Separated Values (legacy tabular format)
- **CUSIP** — Committee on Uniform Securities Identification Procedures (9-character North American security identifier)
- **DOM** — Document Object Model (tree-based XML parsing API)
- **DTD** — Document Type Definition (older XML schema language, predecessor to XSD)
- **EET** — European ESG Template → A.1, Chapter 8

- **EFT** — European Feeder Fund Template → A.1, Chapter 8
- **EMIR** — European Market Infrastructure Regulation
- **EMT** — European MiFID Template → A.1, Chapter 8
- **EPT** — European PRIIPs Template → A.1, Chapter 8
- **ESAP** — European Single Access Point → A.1, Chapter 14
- **ESG** — Environmental, Social, and Governance → A.1
- **ESMA** — European Securities and Markets Authority → A.1
- **ETF** — Exchange-Traded Fund → A.1
- **EU** — European Union
- **FIGI** — Financial Instrument Global Identifier → A.1
- **FinDatEx** — European Working Group on Investment Data Exchange → A.1, Chapter 8
- **FMA** — Finanzmarktaufsicht (Austrian financial market authority)
- **GLEIF** — Global Legal Entity Identifier Foundation → A.1
- **HTTPS** — HyperText Transfer Protocol Secure
- **ISIN** — International Securities Identification Number → A.1
- **ISO** — International Organization for Standardization
- **JAXP** — Java API for XML Processing
- **JSON** — JavaScript Object Notation
- **KID** — Key Information Document → A.1, Chapter 8
- **KVG** — Kapitalverwaltungsgesellschaft (German: management company)
- **LEI** — Legal Entity Identifier → A.1
- **MIC** — Market Identifier Code (ISO 10383, 4-character venue identifier)
- **MiFID II** — Markets in Financial Instruments Directive II → A.1, Chapter 8
- **NAV** — Net Asset Value → A.1, Chapter 5
- **PAI** — Principal Adverse Impacts → A.1, Chapter 8
- **PMS** — Portfolio Management System (internal system type, not a standard acronym)
- **PRIIPs** — Packaged Retail and Insurance-based Investment Products → A.1, Chapter 8
- **RRDM** — Regulatory Reporting Data Mart (internal system type, not a standard acronym)
- **REST** — Representational State Transfer
- **RTS** — Regulatory Technical Standards (issued by ESMA)
- **SAX** — Simple API for XML (streaming XML parser interface)
- **SEDOL** — Stock Exchange Daily Official List (7-character UK/Irish security identifier)
- **SFDR** — Sustainable Finance Disclosure Regulation → A.1, Chapter 8
- **SFTP** — SSH File Transfer Protocol
- **SFTR** — Securities Financing Transactions Regulation
- **StAX** — Streaming API for XML (Java pull-parser)
- **TNAV** — Total Net Asset Value → A.1, Chapter 5

- **TPT** — Tripartite Template → A.1, Chapter 8
- **UAT** — User Acceptance Testing → A.1, Chapter 13
- **UCITS** — Undertakings for Collective Investment in Transferable Securities → A.1, Chapter 1
- **URL** — Uniform Resource Locator
- **UTC** — Coordinated Universal Time
- **UUID** — Universally Unique Identifier
- **WKN** — Wertpapierkennnummer → A.1
- **XML** — Extensible Markup Language → A.1
- **XMLDSig** — XML Digital Signature → A.1, Chapter 9
- **XPath** — XML Path Language → A.1
- **XQuery** — XML Query Language
- **XSD** — XML Schema Definition → A.1
- **XSLT** — Extensible Stylesheet Language Transformations → A.1

PART IV — OUTLOOK AND REFERENCE · APPENDIX B

XML Quick Reference

A compact cheat sheet for XML syntax, XPath, and XSLT

This appendix is a reference, not a tutorial. It assumes you have read Chapter 2 (or already know XML) and need to look up a specific piece of syntax quickly while working on a FundsXML problem. Conventions: syntax fragments are shown in `monospace`, operator and function tables are compact, and worked examples use short FundsXML snippets rather than abstract placeholders. Where XPath or XSLT behaviour differs between version 1.0 and version 2.0, the version is stated explicitly.

B.1 XML Syntax

B.1.1 Document Structure

A well-formed XML document has three parts: an optional XML declaration, an optional prolog (comments, processing instructions, doctype), and exactly one root element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- optional comment -->
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <!-- content -->
</FundsXML4>
```

The XML declaration is optional but strongly recommended. `version` must be `"1.0"` for FundsXML. `encoding` should be `"UTF-8"` unless you have a specific reason to pick another encoding. A third optional attribute, `standalone="yes"`, declares that the document has no external dependencies; FundsXML does not set it.

B.1.2 Elements and Attributes

Form	Syntax
Open + close	<code><Name>content</Name></code>
Self-closing (empty)	<code><Name/></code>
With attributes	<code><Name attr1="value1" attr2="value2">content</Name></code>
Empty with attributes	<code><Name attr="value"/></code>

Element names are case-sensitive. They must start with a letter or underscore, and may contain letters, digits, hyphens, underscores, and periods. Colons are reserved for namespace prefixes.

Attribute values must always be quoted (single or double quotes, consistently matched). Attributes may not appear more than once on the same element.

B.1.3 Text Content and Character Escaping

Five characters have special meaning inside XML content and must be escaped:

Character	Entity	Where required
<	<code>&lt;</code>	Always, in element content and attribute values
>	<code>&gt;</code>	Recommended, always safe
&	<code>&amp;</code>	Always, in element content and attribute values
"	<code>&quot;</code>	Inside double-quoted attribute values
'	<code>&apos;</code>	Inside single-quoted attribute values

Numeric character references are also supported: `©` for © (decimal), `©` for © (hex).

CDATA sections are an alternative for blocks of text containing many special characters. Everything between `<![CDATA[` and `]]>` is treated literally, with no entity processing. Useful for embedded HTML, code samples, or regular expressions.

```
<Description><![CDATA[Returns > 5% if A < B && C]]></Description>
```

B.1.4 Comments and Processing Instructions

```
<!-- a comment, cannot contain -- or end with - -->
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Comments may appear anywhere in the document except inside a tag. Processing instructions are of the form `<?target data?>`; the only one FundsXML practitioners usually encounter is `xml-stylesheet`, which tells a browser to apply an XSLT transformation when rendering.

B.1.5 Namespaces

A namespace is declared with `xmlns` (default namespace) or `xmlns:prefix` (prefixed namespace), and it scopes the element and attribute names below the declaration to a specific vocabulary identified by a URI.

Pattern	Meaning
<code>xmlns="URI"</code>	Elements in scope belong to this default namespace
<code>xmlns:p="URI"</code>	Prefix <code>p:</code> refers to this namespace
<code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>	The XML Schema instance namespace (for <code>xsi:type</code> , <code>xsi:nil</code> , <code>xsi:schemaLocation</code>)
<code>xsi:noNamespaceSchemaLocation="file.xsd"</code>	Locate the schema when there is no target namespace
<code>xsi:schemaLocation="URI file.xsd"</code>	Locate the schema for a specific namespace

FundsXML is defined without a target namespace — it uses `xsi:noNamespaceSchemaLocation`, not `xsi:schemaLocation`. This is unusual for a modern XML standard but deliberate: it keeps element names unprefixes and simplifies XPath expressions.

B.1.6 Well-Formedness Checklist

A document is **well-formed** if and only if:

- Exactly one root element contains all other content.
- Every opening tag has a matching closing tag (or is self-closing).
- Elements are properly nested — no overlapping tags.
- Attribute values are quoted.
- The five special characters are escaped in text content.
- Element and attribute names follow the naming rules.
- Comments and CDATA sections are well-formed.

Well-formedness is a prerequisite for validity. A document can be well-formed without being valid against a schema.

B.2 XPath 1.0 and 2.0

B.2.1 Location Paths — Abbreviated Syntax

XPath expressions select nodes in a document. The most common are **location paths**: a sequence of steps separated by `/`.

Syntax	Meaning
<code>/</code>	Root of the document
<code>/FundsXML4</code>	Root element (absolute path)
<code>FundsXML4</code>	Relative path from the context node
<code>/FundsXML4/ControlData</code>	Child path
<code>//Fund</code>	All <code>Fund</code> elements anywhere in the document
<code>.</code>	Current context node
<code>..</code>	Parent of the context node
<code>@LEI</code>	The <code>LEI</code> attribute of the context node
<code>*</code>	All child elements of the context node
<code>@*</code>	All attributes of the context node
<code>text()</code>	Text content of the context node
<code>node()</code>	Any node (element, text, comment, PI)

B.2.2 Axes

XPath supports thirteen **axes** that control the direction of navigation from the context node. The most-used have abbreviated forms; the full axis syntax is `axis::node-test`.

Axis	Abbreviated	Selects
child::	(default)	Direct children
attribute::	@	Attributes
descendant::	—	All descendants
descendant-or-self::	//	Descendants and the node itself
parent::	..	The parent
ancestor::	—	All ancestors
ancestor-or-self::	—	Ancestors and the node itself
self::	.	The node itself
following-sibling::	—	Siblings appearing later
preceding-sibling::	—	Siblings appearing earlier
following::	—	All later nodes in document order
preceding::	—	All earlier nodes in document order
namespace::	—	Namespace nodes (rarely used)

B.2.3 Predicates

A **predicate** is a filter expression in square brackets that narrows a node-set.

Syntax	Meaning
<code>Fund[1]</code>	The first <code>Fund</code> child
<code>Fund[last()]</code>	The last <code>Fund</code> child
<code>Fund[position() = 2]</code>	The second <code>Fund</code> child
<code>Fund[@LEI]</code>	<code>Fund</code> elements that have a <code>LEI</code> attribute
<code>Fund[@LEI='549300ABCDEFGHJI1234']</code>	<code>Fund</code> elements whose <code>LEI</code> attribute has that value
<code>Fund[Identifiers/LEI]</code>	<code>Fund</code> elements that contain an <code>Identifiers/LEI</code> child
<code>Position[MarketValue > 1000000]</code>	Positions with market value above one million
<code>Fund[Currency='EUR']</code> <code>[SingleFundFlag='true']</code>	Multiple predicates are AND-combined

B.2.4 Operators

Category	Operators
Comparison	<code>=</code> , <code>!=</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code>
Boolean	<code>and</code> , <code>or</code> , <code>not()</code>
Arithmetic	<code>+</code> , <code>-</code> , <code>*</code> , <code>div</code> , <code>mod</code> (note: <code>/</code> is path, not division)
Node-set	<code> </code> (union), <code>intersect</code> , <code>except</code> (XPath 2.0 only)

In XPath 1.0, comparisons of node-sets are existential: `@x > 5` is true if *any* matching node has a value greater than 5. Be aware of this when writing predicates over collections.

B.2.5 Core Functions

Node-set functions:

Function	Returns
<code>count(node-set)</code>	Number of nodes
<code>position()</code>	Position of context node (1-based)
<code>last()</code>	Position of the last node in the current context
<code>name()</code>	Qualified name of the context node
<code>local-name()</code>	Local name without namespace prefix
<code>namespace-uri()</code>	Namespace URI of the context node

String functions:

Function	Returns
<code>string(arg)</code>	String value of the argument
<code>concat(s1, s2, ...)</code>	Concatenation
<code>starts-with(s, prefix)</code>	Boolean
<code>contains(s, substring)</code>	Boolean
<code>substring(s, start, length)</code>	Substring (1-based)
<code>substring-before(s, delim)</code>	Part before the delimiter
<code>substring-after(s, delim)</code>	Part after the delimiter
<code>string-length(s)</code>	Character count
<code>normalize-space(s)</code>	Trims and collapses whitespace
<code>translate(s, from, to)</code>	Character-by-character substitution

Numeric functions:

Function	Returns
<code>number(arg)</code>	Numeric coercion
<code>sum(node-set)</code>	Sum of numeric values
<code>floor(n)</code> , <code>ceiling(n)</code> , <code>round(n)</code>	Rounding

Boolean functions:

Function	Returns
<code>true()</code> , <code>false()</code>	Boolean constants
<code>not(bool)</code>	Logical negation
<code>boolean(arg)</code>	Boolean coercion

XPath 2.0 additions (partial list, for the functions FundsXML practitioners encounter most): `matches(string, regex)` for regular-expression matching; `tokenize(string, regex)` for splitting; `replace(string, regex, replacement)`; `upper-case()`, `lower-case()`; date/time functions `current-date()`, `year-from-date()`, `month-from-date()`, `day-from-date()`; `if (cond) then x else y` conditional expressions; and the `for $var in seq return expr` iteration construct.

B.2.6 FundsXML XPath Recipes**Select a fund by LEI:**

```
//Fund[Identifiers/LEI='549300ABCDEFGHJIJ1234']
```

Get the TNAV amount of that fund:

```
//Fund[Identifiers/LEI='549300ABCDEFGHJIJ1234']
  /FundDynamicData/TotalAssetValues/TotalAssetValue/TotalNetAssetValue/Amount
```

Count all positions across all portfolios of a fund:

```
count(//Fund[Identifiers/LEI='549300ABCDEFGHJIJ1234']//Position)
```

Sum the market values of all equity positions:

```
sum(//Position[PositionType='EQUITY']/TotalValue/Amount)
```

Find all deliveries that are corrections:

```
/FundsXML4[ControlData/DataOperation='AMEND']
```

List all ISINs in a portfolio:

```
//Portfolio/Positions/Position/UniqueID[@IDType='ISIN']
```

Find funds whose name contains "Europa":

```
//Fund[contains(Names/OfficialName, 'Europa')]
```

Select all positions valued above one million euros:

```
//Position[TotalValue/Amount[@ccy='EUR'] > 1000000]
```

B.3 XSLT 1.0 and 2.0

B.3.1 Stylesheet Skeleton

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html" indent="yes" encoding="UTF-8"/>

  <xsl:template match="/">
    <!-- transformation logic -->
  </xsl:template>

</xsl:stylesheet>
```

Change `version="1.0"` to `version="2.0"` to use XSLT 2.0 features (which require an XSLT 2.0 processor such as Saxon). The `xsl:output` element controls serialisation: `method` is `"xml"`, `"html"`, or `"text"`; `indent="yes"` produces human-readable output; `encoding` sets the output encoding.

B.3.2 Template Matching

```
<xsl:template match="pattern">
  <!-- body -->
</xsl:template>
```

Match pattern	Applies to
<code>"/"</code>	The document root
<code>"Fund"</code>	Every <code>Fund</code> element
<code>"Fund/Names/OfficialName"</code>	<code>OfficialName</code> elements with that parent chain
<code>"Fund[@LEI]"</code>	<code>Fund</code> elements with a <code>LEI</code> attribute
<code>"Position[TotalValue/Amount > 1000000]"</code>	Positions above one million
<code>"@*"</code>	Any attribute
<code>"text()"</code>	Any text node
<code>"node()"</code>	Any node

Invocation is either by applying templates (let XSLT dispatch based on `match`) or by calling a named template explicitly:

```
<xsl:apply-templates/>                                <!-- apply to all children -->
<xsl:apply-templates select="Funds/Fund"/> <!-- apply to specific nodes -->
<xsl:call-template name="fund-summary"/>  <!-- call a named template -->
```

B.3.3 Selecting and Copying Values

Instruction	Purpose
<code><xsl:value-of select="expr"/></code>	Output the string value of an expression
<code><xsl:copy-of select="expr"/></code>	Deep-copy nodes (preserving structure)
<code><xsl:copy/></code>	Shallow-copy the context node (useful in identity transforms)
<code><xsl:text>literal</xsl:text></code>	Output literal text with controlled whitespace

B.3.4 Iteration and Conditionals

Iteration:

```
<xsl:for-each select="//Fund">
  <li><xsl:value-of select="Names/OfficialName"/></li>
</xsl:for-each>
```

Simple condition:

```
<xsl:if test="Currency='EUR'">
  <p>This is a euro-denominated fund.</p>
</xsl:if>
```

Multi-branch condition:

```
<xsl:choose>
  <xsl:when test="SingleFundFlag='true'">Standalone fund</xsl:when>
  <xsl:when test="SingleFundFlag='false'">Sub-fund of an umbrella</xsl:when>
  <xsl:otherwise>Unknown structure</xsl:otherwise>
</xsl:choose>
```

Sorting inside an iteration:

```
<xsl:for-each select="//Fund">
  <xsl:sort select="Names/OfficialName" order="ascending"/>
  <li><xsl:value-of select="Names/OfficialName"/></li>
</xsl:for-each>
```

B.3.5 Variables and Parameters

```
<xsl:variable name="fund-lei" select="'549300ABCDEFGHJIJ1234'"/>
<xsl:variable name="fund-count" select="count(//Fund)"/>

<xsl:param name="report-date" select="'2026-03-31'"/>

<xsl:value-of select="$fund-lei"/>
```

Variables are bound once and cannot be reassigned. Parameters behave like variables but can be passed into templates via `<xsl:with-param name="..." select="..." />`.

B.3.6 FundsXML XSLT Recipes

Recipe 1 — Extract fund NAVs to an HTML table.

```
<xsl:template match="/">
  <html><body>
    <h1>Fund NAV Summary</h1>
    <table border="1">
      <tr><th>Fund</th><th>LEI</th><th>NAV</th><th>Currency</th></tr>
      <xsl:for-each select="//Fund">
        <tr>
          <td><xsl:value-of select="Names/OfficialName"/></td>
          <td><xsl:value-of select="Identifiers/LEI"/></td>
```

```

        <td><xsl:value-of select=
            "FundDynamicData/TotalAssetValues/TotalAssetValue
            /TotalNetAssetValue/Amount"/></td>
        <td><xsl:value-of select="Currency"/></td>
    </tr>
</xsl:for-each>
</table>
</body></html>
</xsl:template>

```

Recipe 2 — Identity transform with one exception (strip `RegulatoryReportings`).

```

<!-- copy everything by default -->
<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>

<!-- but drop RegulatoryReportings blocks -->
<xsl:template match="RegulatoryReportings"/>

```

This is the standard pattern for a surgical transformation: a general identity template plus a small number of overriding templates that handle the specific cases you want to change.

Recipe 3 — Sum all position values in a portfolio.

```

<xsl:template match="Portfolio">
    <p>
        Portfolio total:
        <xsl:value-of select="sum(Positions/Position/TotalValue/Amount)"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="Positions/Position[1]/TotalValue/Amount/@ccy"/>
    </p>
</xsl:template>

```

The `<xsl:text> </xsl:text>` is an explicit single space — whitespace in the stylesheet source is not preserved unless wrapped in `xsl:text`, so forcing a space this way is the reliable idiom.

PART IV — OUTLOOK AND REFERENCE · APPENDIX C

FundsXML Schema Overview

Structured reference for elements, types, and enumerations

This appendix is a structured reference to the FundsXML 4.2 schema (version 4.2.8 as of the time of writing). Chapters 4 through 8 introduced the same material narratively and with deliberate pedagogical simplifications; this appendix gives the authoritative structural view. Where the narrative chapters presented simplified element names or paths for teaching purposes, the tables below give the real schema names, types, cardinalities, and enumeration values.

Scope. The schema defines well over six hundred complex types. Most of those exist to model edge cases — specific corporate actions, niche asset classes, country-specific disclosures — and are rarely touched by a typical producer pipeline. This appendix covers the **curated subset** that every producer and consumer implementation encounters: the top-level structure, the `ControlData` envelope, the core `Fund` and `FundStaticData`/`FundDynamicData` blocks, the `Portfolio` and `Position` model, the `Transactions` model, the `RegulatoryReportings` block, the common reusable types, and the extension mechanisms. A full schema browser (the `FreeXmlToolkit` XSD view from Chapter 11, or an online schema viewer) should be consulted for anything beyond this subset.

Conventions.

- **Cardinality** is given as `1..1` (required, exactly one), `0..1` (optional), `0..n` (any number including zero), or `1..n` (at least one).
- **Type** is the schema type name where the element references a named type; simple built-in types (`xs:date`, `xs:decimal`, `xs:boolean`, `xs:dateTime`, `xs:string`, etc.) are shown with the `xs:` prefix.
- **■** marks enumerated values. For short enumerations the full list is shown inline; for longer ones the list is given in §C.7.
- **⊕** marks extension points (places where `CustomAttributes` or `CountrySpecificData` may be used).
- All element and type names are **case-sensitive** exactly as written. FundsXML does not use a target namespace, so the names below appear unprefixed in instance documents.

C.1 Top-Level Structure

The root element is `FundsXML4`. Its children form the main content blocks of the document. `ControlData` is mandatory; everything else is optional, and a valid document may contain only `ControlData` together with one or more of the optional blocks appropriate to the delivery's purpose.

Element	Type	Cardinality	Purpose
ControlData	ControlDataType	1..1	Envelope metadata — document ID, date, supplier, operation. See §C.2.
Funds	(inline complex)	0..1	Container for Fund elements. See §C.3.
AssetMgmtCompanyDynData	AssetMgmtCompanyDynDataType	0..1	Dynamic data of asset management companies (sales, AuM, statistics).
AssetMasterData	AssetMasterDataType	0..1	Reference data for instruments referenced by portfolio positions. See §C.4.4.
Documents	DocumentsType	0..1	References to fund-related documents (KID, prospectus, annual report). See §C.7.5.
RegulatoryReportings	RegulatoryReportingsType	0..1	FinDatEx templates (EMT, EPT, EET, EFT, TPT). See §C.6.
CountrySpecificData	CountrySpecificDataType	0..1	Country-specific extensions at the document level. ☸
ds:Signature	XMLDSig	0..1	Enveloped XML Digital Signature. See Chapter 9.

Note one detail that is occasionally surprising: `RegulatoryReportings` lives at the **document root**, not inside each `Fund`. A delivery that covers multiple funds with regulatory reporting data places the reporting block once at the top level, with internal references tying each report to the relevant fund.

C.2 ControlData Block

`ControlData` is the envelope metadata of every FundsXML delivery. Every consumer inspects `ControlData` first to decide how to handle the incoming document: who sent it, when it was generated, what operation it represents, and which earlier delivery (if any) it relates to.

C.2.1 Direct Children of ControlData

Element	Type	Cardinality	Purpose
UniqueDocumentID	Text128Type	1..1	Document identifier, unique per data supplier.
DocumentGenerated	xs:dateTime	1..1	Timestamp at which the document was created.
Version	(enum)	0..1	Schema version: 4.0.0 through 4.2.8 .
ContentDate	xs:date	1..1	Main valuation date for the delivery.
DataSupplier	DataSupplierType	1..1	Organisation that produced the document. See §C.2.2.
DataOperation	(enum)	0..1	INITIAL , AMEND , or DELETE .
RelatedDocumentIDs	(inline)	0..1	Back-references to related deliveries. Required for DELETE .
Language	xs:language	0..1	Default language for text content.
ModuleUsage	(inline)	0..1	List of schema modules used. See §C.2.3.
CountrySpecificData	(inline)	0..1	Country-specific ControlData sections.
CustomAttributes	AttributesType	0..1	Producer-specific extension data.

C.2.2 DataSupplierType

Element	Type	Cardinality	Purpose
SystemCountry	ISOCountryCodeType	1..1	Two-character ISO country code.
Short	Text64Type	1..1	Short code of supplier (unique per country).
Name	Text256Type	1..1	Legal name of data supplier.
Type	Text64Type	1..1	Type of supplier: IC (investment company), CB (central bank), NB (national bank), Vendor , etc.
Contact	(inline)	0..1	Optional contact block (Name, Phone, Email).

Note that `DataSupplierType` does **not** contain an `LEI` field directly; the supplier is identified by the country-scoped `Short` code, with the legal name as a human-readable label.

C.2.3 ModuleUsage Enumeration

`ControlData/ModuleUsage/Module/Name` allows the producer to declare which schema modules the delivery exercises, so that consumers can skip parsing blocks that are not present. Valid values are:

```
AssetMasterData · AssetMgmtCompDynData · CountrySpecificData_AT ·
CountrySpecificData_DE · CountrySpecificData_DK · CountrySpecificData_FR ·
CountrySpecificData_LU · CountrySpecificData_NL · FundDynamicData · FundStaticData ·
PortfolioData · RegulatoryReporting_EMIR · RegulatoryReporting_EMT ·
RegulatoryReporting_KIID · RegulatoryReporting_PRIIPS · RegulatoryReporting_SolvencyII
· ShareClassData · TransactionData
```

C.2.4 RelatedDocumentIDs

```
RelatedDocumentIDs
└─ RelatedDocumentID (Text128Type) – 1..n
```

A single `RelatedDocumentIDs` element contains one or more `RelatedDocumentID` children, each of which is a `UniqueDocumentID` from a previous delivery. `DELETE` operations must specify which delivery is being retracted via this element; `AMEND` operations may optionally specify which delivery they supersede.

C.3 Fund Block

`Funds` is a container with exactly one child element: `Fund`, which appears once per fund described by the delivery.

```
Funds
└─ Fund (FundType) – 1..n
```

C.3.1 FundType — Direct Children

Element	Type	Cardinality	Purpose
Identifiers	IdentifiersType	1..1	Fund identifiers (LEI, ISIN, etc.). See §C.7.2.
Names	NamesType	1..1	Fund names. See §C.3.2.
Currency	ISOCurrencyCodeType	1..1	Base currency (ISO 4217).
SingleFundFlag	xs:boolean	1..1	true for standalone funds, false for umbrella structures.
DataSupplier	DataSupplierType	0..1	Fund-level supplier, if different from ControlData.
FundStaticData	FundStaticDataType	0..1	Rarely-changing fund metadata. See §C.3.3.
FundDynamicData	FundDynamicDataType	0..1	Valuation-date data. See §C.3.4.
SingleFund or Subfunds	(choice)	0..1	Choice between SingleFundType (standalone) and a Subfunds container holding one or more SubfundType .
CountrySpecificData	CountrySpecificFundDataType	0..1	Country-specific fund extensions. ☸

The `SingleFund` / `Subfunds` choice reflects the umbrella-structure distinction: a standalone fund uses `SingleFund` (which holds `ShareClasses` and optional `Segments`); an umbrella fund uses `Subfunds` (which holds one or more `Subfund` elements, each with its own share classes and segments).

C.3.2 NamesType

Element	Type	Cardinality	Purpose
OfficialName	Text500Type	1..1	Legal official name.
FullName	Text500Type	0..1	Full name (for sub-funds within an umbrella).
MarketingName	Text500Type	0..1	Marketing name.
ShortName	Text100Type	0..1	Short name (≤ 100 characters).
PreviousName	(inline)	0..1	Previous name with effective date range.

C.3.3 FundStaticDataType — Major Children

`FundStaticData` is large (roughly twenty direct children plus nested structures). The elements a typical producer populates are:

Element	Type	Cardinality	Purpose
DomicileCountry	ISOCountryCodeType	0..1	Two-character ISO country code.
ListedLegalStructure / UnlistedLegalStructure	(choice)	0..1	Either an enumerated legal structure or free text. See §C.7.4.
InceptionDate	xs:date	0..1	Date the fund was launched.
StartOfFiscalYear	DayMonthType	0..1	First day of fiscal year (day/month pair).
EndOfFiscalYear	DayMonthType	0..1	Last day of fiscal year (day/month pair).
OpenClosedEnded	(enum)	0..1	OPEN or CLOSED . ▶
MaturityDate	xs:date	0..1	Planned end date (for closed-ended funds).
LiquidationDate	xs:date	0..1	Date the fund was closed.
Administrator	CompanyType	0..1	Fund administrator.
Auditor	CompanyType	0..1	External auditor.
Custodian	CompanyType	0..1	Depository / custodian bank.
InvestmentCompany	CompanyType	0..1	Asset management company.
FundTexts	(inline)	0..1	Multi-language marketing and descriptive text.
Classifications	ClassificationsType	0..1	Classification schemes (BVI, SRI, etc.).
Benchmarks	(inline)	0..1	Benchmark static data.
FundHedgingStrategy	HedgingStrategyType	0..1	Hedging approach.
OngoingCosts	(inline)	0..1	Fee structure (management fee, performance fee, transaction costs).
SFDRProductType	(enum)	0..1	0 , 6 , 8 , or 9 . ▶ See §C.7.3.
CustomAttributes	AttributesType	0..1	Extension data. ⊕

C.3.4 FundDynamicDataType

`FundDynamicData` is surprisingly compact — it has only three direct children, each of which contains the real complexity:

Element	Type	Cardinality	Purpose
<code>TotalAssetValues</code>	(inline)	0..1	Container for <code>TotalAssetValue</code> elements (one per NAV date / variant).
<code>Portfolios</code>	(inline)	0..1	Container for <code>Portfolio</code> elements (one per valuation date).
<code>Benchmarks</code>	(inline)	0..1	Container for <code>Benchmark</code> dynamic-data elements.

C.3.5 TotalAssetValueType

Element	Type	Cardinality	Purpose
<code>NavDate</code>	<code>xs:date</code>	1..1	Valuation date.
<code>TotalAssetNature</code>	(enum)	1..1	OFFICIAL , ESTIMATED , or TECHNICAL . 
<code>TotalNetAssetValue</code>	<code>FundAmountType</code>	0..1	Total net asset value (TNAV).
<code>TotalGrossAssetValue</code>	<code>FundAmountType</code>	0..1	Gross asset value (closed-ended and real-estate funds).
<code>SharesOutstanding</code>	<code>xs:decimal</code>	0..1	Number of shares used for NAV calculation.
<code>Ratio</code>	<code>PercentageType</code>	0..1	Share-class weight within the fund (sums to 100 %).
<code>OtherTotalAssetValues</code>	(inline)	0..1	Additional variants (swing NAV, hold-to-maturity, etc.).

C.3.6 SingleFundType

Element	Type	Cardinality	Purpose
<code>SingleFundStaticData</code>	<code>SingleFundStaticDataType</code>	0..1	Static data specific to the standalone fund structure.
<code>ShareClasses</code>	(inline)	0..1	Container for <code>ShareClass</code> elements.
<code>Segments</code>	(inline)	0..1	Container for <code>Segment</code> elements (for funds internally split by manager).

C.3.7 ShareClassType — Major Children

A share class has its own identifiers, names, currency, and a detailed set of dynamic data. The major children are:

Element	Type	Cardinality	Purpose
Identifiers	IdentifiersType	1..1	Share-class identifiers (ISIN, WKN, etc.).
Names	NamesType	1..1	Share-class names.
Currency	ISOCurrencyCodeType	1..1	Share-class currency.
ShareClassType	ShareClassTypeType	0..1	Share-class type designation (A, B, I, etc.).
InceptionDate	xs:date	0..1	Date the share class was launched.
LiquidationDate	xs:date	0..1	Date the share class was closed.
StaticData	(inline)	0..1	Share-class-specific static details.
DynamicData	(inline)	0..1	NAV, prices, flows, earnings.

C.4 Portfolio Block

The portfolio block lives inside `FundDynamicData/Portfolios`. Each `Portfolio` element represents a snapshot of holdings at a specific valuation date.

C.4.1 PortfolioType

Element	Type	Cardinality	Purpose
NavDate	xs:date	1..1	Valuation date for all data in this portfolio block.
Positions	PositionsType	0..1	Holdings (securities, cash, accounts, fees).
Transactions	TransactionsType	0..1	Transactions within the portfolio. See §C.5.
Earnings	EarningsType (extended)	0..1	Earnings data (coupons, dividends, distributions).
PositionsDecomposed	PositionsDecomposedType	0..1	Look-through positions (underlying holdings of fund-of-fund positions).
BreakDowns	BreakDownsType	0..1	Aggregated breakdown statistics (by country, sector, currency).

C.4.2 PositionType — Major Children

Element	Type	Cardinality	Purpose
UniqueID	xs:IDREF	1..1	Reference to AssetMasterData/Asset/UniqueID .
Identifiers	IdentifiersType	0..1	Duplicate identifiers for convenience (ISIN, ticker).
Currency	ISOCurrencyCodeType	0..1	Valuation currency of the position.
TotalValue	FundAmountType	1..1	Total position value (sum across all positions equals fund TNAV).
OtherTotalValues	(inline)	0..1	Additional valuations (e.g. hold-to-maturity).
Units	xs:decimal	0..1	Number of units or nominal amount.
PriceValue	AmountType	0..1	Unit price at valuation.
PricePercent	xs:decimal	0..1	Price as a percentage of nominal (for bonds).
AccruedInterest	AmountType	0..1	Accrued interest amount (for interest-bearing instruments).
Rates	(inline)	0..1	FX rates used for currency translation.
Account	AccountType	0..1	Cash-account details (for account positions).
Equity , Bond , Warrant , ...	(inline)	0..1 each	Instrument-type-specific position details.

C.4.3 Position Instrument Coverage

A `Position` element carries instrument-type-specific data in one of several optional sub-elements. The sub-element used depends on the `AssetType` of the referenced asset in `AssetMasterData` . The most-used sub-elements are:

Sub-element	Covers
Equity	Equities, ordinary and preference shares.
Bond	Government bonds, corporate bonds, convertible bonds (basic fields).
ConvertibleBond	Convertible bond details (conversion dates, price, ratio).
Warrant	Warrants.
Option	Options (calls, puts, American, European).
Future	Futures contracts.
FXForward	FX forward contracts.
Swap	Interest-rate, currency, CDS, and total-return swaps.
Repo	Repo and reverse-repo contracts.
FixedTimeDeposit	Term deposits.
RealEstate	Direct real-estate holdings.
REIT	Real estate investment trusts.
Fund	Holdings of other funds (including ETFs).
PrivateEquity	Private-equity participations.
Certificate	Certificates and structured products.
Commodity	Commodity holdings.
Crypto	Cryptocurrency holdings.

C.4.4 AssetMasterData and AssetType

`AssetMasterData/Asset` is the reference data block. Each asset carries a `UniqueID` (schema type `xs:ID`) that position elements reference via `xs:IDREF`.

Element	Type	Cardinality	Purpose
UniqueID	xs:ID	1..1	Document-unique identifier (target of IDREF from positions).
Identifiers	IdentifiersType	0..1	ISIN, ticker, etc.
Currency	ISOCurrencyCodeType	1..1	Nominal currency of the instrument.
Country	ISOCountryCodeType	0..1	Country of the issuer or instrument.
Name	Text256Type	1..1	Human-readable name.
AssetType	(enum)	1..1	Two-character asset-type code. See below. 
AssetDetails	AssetDetailsType	0..1	Type-specific details.
Ratings	(inline)	0..1	Credit ratings.
CountrySpecificData	(inline)	0..1	Country-specific asset extensions. 

The `AssetType` enumeration uses two-character codes:

Code	Meaning	Code	Meaning
EQ	Equity	AC	Account
B0	Bond (incl. convertibles)	FE	Fee
SC	Share Class (fund unit)	RE	Real Estate
WA	Warrant	RT	REIT
CE	Certificate	L0	Loan
OP	Option	RI	Right
FU	Future	C0	Commodity
FX	FX-Forward	PE	Private Equity
SW	Swap	CP	Commercial Paper
RP	Repo	IX	Index
FT	Fixed Time Deposit	CR	Crypto
CM	Call Money		

C.5 Transactions

`Portfolio/Transactions` contains a list of `Transaction` elements. Each transaction has a unique `TransactionID` (enforced by a schema-level key constraint on the container) and one of five possible kinds.

C.5.1 TransactionType — Major Children

Element	Type	Cardinality	Purpose
TransactionID	Text256Type	1..1	Unique transaction identifier.
CancellationFlag	xs:boolean	0..1	true if the transaction has been cancelled.
OriginalTransactionID	Text256Type	0..1	For cancellations, the ID of the original transaction.
AssetUniqueID	xs:IDREF	0..1	Reference to AssetMasterData/Asset/UniqueID .
Identifiers	IdentifiersType	0..1	Duplicate identifiers of the instrument.
Currency	ISOCurrencyCodeType	0..1	Instrument currency.
TransactionKind	(enum)	1..1	BUY , SELL , LENDING_BUY , LENDING_SELL , or CORP_ACTION . ▶
SettlementCurrency	ISOCurrencyCodeType	0..1	Settlement currency (may differ from instrument currency).
EntryDate	xs:date	0..1	Booking date.
ValutaDate	xs:date	0..1	Value date (effective date).
ClosingDate	xs:date	0..1	Trade date.
OrderTimestamp	xs:dateTime	0..1	Exact order time.
(numerous amount and quantity fields)			Quantity, price, gross/net values, commissions, taxes, FX rates.

Note that the `TransactionKind` enumeration has exactly five values. The narrative in Chapter 7 used pedagogical labels (`SUBSCRIPTION` , `REDEMPTION` , etc.) that do not appear in the real schema — investor flows are modelled at the share-class level, not as portfolio transactions. The schema-accurate distinction is: **Transaction covers portfolio trading activity; investor flows are captured elsewhere in the share-class dynamic data.**

C.5.2 EarningsType

`Portfolio/Earnings` carries coupons, dividends, and distributions resulting from portfolio positions. The block is attributed with `from` and `to` dates that bound the reporting period. The underlying `EarningsType` carries per-position earnings entries with amounts, dates, and tax treatment; the structure is too detailed for this overview — consult the schema browser if you need to populate it.

C.6 RegulatoryReportings and FinDatEx Templates

`RegulatoryReportings` at the document root has exactly two children:

Element	Type	Cardinality	Purpose
<code>DirectReporting</code>	<code>DirectReportingType</code>	0..1	Direct regulatory reporting (EMIR, EMT, EFT, EET, KIIDs).
<code>IndirectReporting</code>	<code>IndirectReportingType</code>	0..1	Indirect reporting for downstream consumers (TPT, PRIIPS/EPT).

The terminology reflects the distinction between reports that flow directly to regulators (direct) and reports that flow through distributors or insurance-company intermediaries (indirect).

C.6.1 DirectReporting Children

Element	Type	Cardinality	Purpose
<code>EMIR</code>	<code>EMIRType</code>	0..1	OTC derivatives reporting per Regulation (EU) 648/2012.
<code>KIIDs</code>	<code>KIIDsType</code>	0..1	UCITS KIID data (predecessor of PRIIPs KID).
<code>EMT</code>	<code>EMTType</code>	0..1	EMT v1.0 (2017).
<code>EMT_V3</code>	<code>EMTType_V3</code>	0..1	EMT v3.0 (2019).
<code>EMT_V4</code>	<code>EMTType_V4</code>	0..1	EMT v4.0 (2022).
<code>EMT_V4_1</code>	<code>EMTType_V41</code>	0..1	EMT v4.1 (2023).
<code>EMT_V4_2</code>	<code>EMTType_V42</code>	0..1	EMT v4.2 (2024).
<code>EFTs</code>	<code>EFTType</code>	0..1	European Feedback Template v1.0.
<code>EET1.0</code>	<code>EETsType</code>	0..1	EET v1.0 (2022).
<code>EET1.1</code>	<code>EETsType1.1</code>	0..1	EET v1.1.
<code>EET1.1.1</code>	<code>EETsType1.1.1</code>	0..1	EET v1.1.1.
<code>EET1.1.2</code>	<code>EETsType1.1.2</code>	0..1	EET v1.1.2.
<code>EET1.1.3</code>	<code>EETsType1.1.3</code>	0..1	EET v1.1.3.

Each template version is a distinct element with its own type. A producer emits one of the EMT versions and one of the EET versions — typically the most recent — and consumers accept whichever versions their systems support. The multiple versions enable the gradual cross-ecosystem upgrade cycle that Chapter 13 described.

C.6.2 IndirectReporting Children

Element	Type	Cardinality	Purpose
TripartiteTemplateSolvencyII	TripartiteTemplateSolvencyIIType	0..1	TPT v4 (2018).
TripartiteTemplateSolvencyII_V5	TripartiteTemplateSolvencyIIType_V5	0..1	TPT v5 (2020).
TripartiteTemplateSolvencyII_V6	TripartiteTemplateSolvencyIIType_V6	0..1	TPT v6 (2022).
PRIIPS	PRIIPSType	0..1	EPT v1.0 / v1.1 (PRIIPs template).
PRIIPS_V20	PRIIPSType_V20	0..1	EPT v2.0 and Comfort-EPT v2.0.

The PRIIPS element models both the EPT (European PRIIPs Template) used by distributors and the CEPT (Comfort EPT) used for review before publication. Newer EPT versions live in their own elements as they are published.

C.6.3 Template Coverage

The internal structure of each FinDatEx template — the specific fields, enumerations, and value ranges — is published by FinDatEx and embedded in FundsXML. Each template has between 60 and 180 individual fields, far too many to enumerate here. Chapter 8 introduced the purpose and high-level structure of each template; for field-level detail, consult the FinDatEx specification for the relevant template version.

C.7 Common Types and Enumerations

C.7.1 AmountType and FundAmountType

```

AmountType
├── Amount (xs:decimal) – 1..n
│   └── @ccy (ISOCurrencyCodeType, required) – the currency code

```

AmountType can carry the same amount in multiple currencies (one **Amount** per currency, distinguished by the **ccy** attribute). **FundAmountType** is a variant used within fund- and portfolio-level values where the amount always includes one representation in the fund's base currency.

C.7.2 IdentifiersType

The **IdentifiersType** is used wherever a fund, share class, or instrument is identified. The major children are:

Element	Type	Cardinality	Purpose
ISIN	ISINType	0..1	12-character ISIN.
Bloomberg	(inline)	0..1	Bloomberg Ticker, Exchange, Market, Yellow Key.
Reuters	(inline)	0..1	Reuters identifier.
FIGI	Text32Type	0..1	Financial Instrument Global Identifier.
CFI	Text6Type	0..1	Classification of Financial Instruments (ISO 10962).
LEI	LEICodeType	0..1	Legal Entity Identifier (20 characters, ISO 17442).
WKN	Text6Type	0..1	German Wertpapierkennnummer.
SEDOL	Text7Type	0..1	UK/Irish Stock Exchange Daily Official List code.
CUSIP	Text9Type	0..1	North American identifier.
OtherID	(inline)	0..n	Container for proprietary identifiers.

All identifier fields are optional; at least one is typically populated.

C.7.3 Core Enumerations

DataOperation (in **ControlData/DataOperation**):

Value	Meaning
INITIAL	First-time delivery of new data.
AMEND	Update or correction to previously delivered data.
DELETE	Retract a previous delivery. Requires RelatedDocumentIDs .

TotalAssetNature (in **TotalAssetValue/TotalAssetNature**):

Value	Meaning
OFFICIAL	Final NAV that investors will transact against.
ESTIMATED	Preliminary NAV, pending finalisation.
TECHNICAL	Internal technical value, not for external use.

TransactionKind (in Transaction/TransactionKind):

Value	Meaning
BUY	Purchase of an asset.
SELL	Sale of an asset.
LENDING_BUY	Securities-lending purchase.
LENDING_SELL	Securities-lending sale.
CORP_ACTION	Corporate-action-driven position change.

OpenClosedEnded (in FundStaticData/OpenClosedEnded):

Value	Meaning
OPEN	Open-ended fund (ongoing subscription/redemption).
CLOSED	Closed-ended fund (fixed capital).

SFDRProductType (in FundStaticData/SFDRProductType):

Value	Meaning
0	Unclassified or not applicable.
6	Article 6 product (no sustainability characteristics).
8	Article 8 product (promotes ESG characteristics).
9	Article 9 product (has sustainable investment objective).

C.7.4 Listed Legal Structure Enumeration

FundStaticData/ListedLegalStructure uses these values:

UCITS · UCITS - SICAV · UCITS - SICAF · UCITS - CONTRACTUAL TYPE · AIF · AIF - HEDGE FUND · AIF - PRIVATE EQUITY FUND · AIF - VENTURE CAPITAL FUND · AIF - REAL ESTATE FUND · AIF - REIT · AIF - INFRASTRUCTURE FUND · AIF - COMMODITY FUND · AIF - SOVEREIGN WEALTH FUND · AIF - ELTIF · AIF - EUVECA · AIF - EUSEF · SPV

If none of these fits, use `UnlistedLegalStructure` with free text.

C.7.5 Documents Block

`Documents/Document` represents references to (or embedded content of) fund-related documents. The `Type/ListedType` enumeration is:

Value
AIFMD
AnnualReport
AuditReport
Factsheet
KID
Prospectus
PRIIPS-KID

Other document types are supported via `Type/UnlistedType` (free text). Documents are typically referenced by URL; inline content is also supported but uncommon.

C.7.6 Reusable Text Types

The schema defines several `Text...Type` simple types that restrict element content by length:

Type	Max length
Text16Type	16 characters
Text32Type	32 characters
Text64Type	64 characters
Text100Type	100 characters
Text128Type	128 characters
Text200Type	200 characters
Text256Type	256 characters
Text500Type	500 characters

C.8 Extension Points

C.8.1 CustomAttributes

The `CustomAttributes` element uses `AttributesType`, the standard extension mechanism for producer-specific data that does not fit into any standard element. It is permitted inside `ControlData`, `FundStaticData`, and many other places throughout the schema.

```

CustomAttributes (AttributesType)
├─ Attribute – 1..n
│   ├── Name (Text256Type) – 1..1
│   ├── Type (enum T/N/D/B) – 1..1
│   └─ (choice) – 1..1
│       ├── ValueText (xs:string)
│       ├── ValueNumber (xs:decimal)
│       ├── ValueDate (xs:date)
│       └─ ValueBoolean (xs:boolean)

```

The `Type` indicator and the chosen `Value*` element must agree: `Type="T"` pairs with `ValueText`, `Type="N"` with `ValueNumber`, `Type="D"` with `ValueDate`, `Type="B"` with `ValueBoolean`.

Use `CustomAttributes` sparingly. The first question before adding one should always be whether a standard element already exists; the second question should be whether the data belongs in the delivery at all. Permanent project conventions for naming the `Name` field (for example a `{producer}.{domain}.{fieldname}` scheme) prevent collisions between projects.

C.8.2 CountrySpecificData

`CountrySpecificData` appears at several levels of the schema and allows country-specific extensions structured by ISO country code. The main locations are:

Location	Type
<code>FundsXML4/CountrySpecificData</code>	Root-level country extensions.
<code>FundsXML4/ControlData/CountrySpecificData</code>	Country-specific ControlData.
<code>Funds/Fund/CountrySpecificData</code>	Fund-level country extensions.
<code>AssetMasterData/Asset/CountrySpecificData</code>	Asset-level country extensions.

Within each of these, the sub-elements are named by ISO country code (`AT` , `DE` , `DK` , `FR` , `LU` , `NL` , and others as they are added). Each country sub-element has its own type that defines what country-specific structure is permitted. Country extensions are typically used for regulatory disclosures specific to one distribution country that do not fit into the FinDatEx templates.

C.9 Schema Version Notes

This appendix documents FundsXML schema version **4.2.8**, the version current as of the book's writing in early 2026. The 4.2 minor-version series has been stable since 2019 and is backward-compatible within the series: a document valid against 4.2.0 is also valid against 4.2.8 unless it exercises an element that has been deprecated between the two versions. Producers upgrading between minor versions should consult the changelog published by the FundsXML association for the specific release they are moving to.

Earlier major versions (3.x and prior) are no longer maintained; the 4.0 transition was made many years ago and most elements in the book's target population predate the transition. Anticipated 2026 releases (4.2.9 is expected in November 2026, per the ongoing release cadence) are likely to add new fields to EET and EMT as FinDatEx publishes updated templates, but the structural overview presented in this appendix is expected to remain accurate for the foreseeable future.

PART IV — OUTLOOK AND REFERENCE · APPENDIX D

Complete Example Files

Complete, schema-valid FundsXML sample files for the Europa Growth Fund

This appendix contains four complete FundsXML documents for the fictional Europa Growth Fund, progressing from minimal to production-grade. Every document in this appendix has been **validated against the real FundsXML4.xsd (version 4.2.8)** using `xmlLint --schema` and passes without errors. A reader can copy any of these documents into a file, place it alongside the schema, and reproduce the validation result.

The four documents are:

- **D.1 — Minimal Delivery.** The smallest possible valid FundsXML document: [ControlData](#), one Fund with identifiers, names, currency, and a single TNAV. Start here to understand the bare structure.
- **D.2 — Monthly NAV Delivery.** A mid-sized delivery with [FundStaticData](#), a portfolio containing four positions (two equities, one bond, one cash account), and the matching [AssetMasterData](#) block.
- **D.3 — Full Production Delivery.** A realistic producer output: extended FundStaticData with Custodian, Administrator, Auditor, OngoingCosts, and SFDRProductType; eight portfolio positions including an FX forward; two transactions; three share classes; two document references; and [CustomAttributes](#) in ControlData.
- **D.4 — AMEND Delivery.** A correction delivery that references D.2 via RelatedDocumentIDs, demonstrating the AMEND data operation with a corrected TNAV.

What is not included. The [RegulatoryReportings](#) block (EMT, EPT, EET, EFT, TPT) is omitted because each FinDatEx template contains 60–180 fields that would consume many pages without adding structural insight beyond what Chapter 8 already provides. XML Digital Signatures are omitted because they require real cryptographic keys and cannot be meaningfully reproduced from a book listing. Readers who need RegulatoryReportings or Signature examples should consult the FundsXML association’s sample-file repository.

D.1 Minimal Delivery

The simplest valid FundsXML document: ControlData with all required fields, and a single Fund carrying only identifiers, names, currency, the SingleFundFlag, and one TotalAssetValue. This is the baseline that every more complex delivery builds upon.

```

<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-001</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T06:47:13Z</DocumentGenerated>
  
```

```

<Version>4.2.8</Version>
<ContentDate>2026-03-31</ContentDate>
<DataSupplier>
  <SystemCountry>LU</SystemCountry>
  <Short>EAM</Short>
  <Name>Europa Asset Management S.A.</Name>
  <Type>IC</Type>
</DataSupplier>
<DataOperation>INITIAL</DataOperation>
<Language>en</Language>
</ControlData>
<Funds>
  <Fund>
    <Identifiers>
      <LEI>549300ABCDEFGHJIJ1234</LEI>
    </Identifiers>
    <Names>
      <OfficialName>Europa Growth Fund</OfficialName>
    </Names>
    <Currency>EUR</Currency>
    <SingleFundFlag>true</SingleFundFlag>
    <FundDynamicData>
      <TotalAssetValues>
        <TotalAssetValue>
          <NavDate>2026-03-31</NavDate>
          <TotalAssetNature>OFFICIAL</TotalAssetNature>
          <TotalNetAssetValue>
            <Amount ccy="EUR">464552848.78</Amount>
          </TotalNetAssetValue>
        </TotalAssetValue>
      </TotalAssetValues>
    </FundDynamicData>
  </Fund>
</Funds>
</FundsXML4>

```

D.2 Monthly NAV Delivery

A delivery that a typical producer might emit at month-end: **ControlData**, **FundStaticData** with domicile, legal structure, inception date, fiscal-year boundaries, custodian, investment company, and SFDR classification; **FundDynamicData** with a **TotalAssetValue** and a portfolio of four positions; and the matching **AssetMasterData** block. Each position uses the `UniqueID / xs:IDREF` mechanism to reference its asset in **AssetMasterData**.

```

<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">

```

```

<ControlData>
  <UniqueDocumentID>EGF-20260331-002</UniqueDocumentID>
  <DocumentGenerated>2026-04-01T07:15:00Z</DocumentGenerated>
  <Version>4.2.8</Version>
  <ContentDate>2026-03-31</ContentDate>
  <DataSupplier>
    <SystemCountry>LU</SystemCountry>
    <Short>EAM</Short>
    <Name>Europa Asset Management S.A.</Name>
    <Type>IC</Type>
  </DataSupplier>
  <DataOperation>INITIAL</DataOperation>
  <Language>en</Language>
</ControlData>
<Funds>
  <Fund>
    <Identifiers>
      <LEI>549300ABCDEFGHIIJ1234</LEI>
    </Identifiers>
    <Names>
      <OfficialName>Europa Growth Fund</OfficialName>
      <ShortName>EGF</ShortName>
    </Names>
    <Currency>EUR</Currency>
    <SingleFundFlag>true</SingleFundFlag>
    <FundStaticData>
      <DomicileCountry>LU</DomicileCountry>
      <ListedLegalStructure>UCITS</ListedLegalStructure>
      <InceptionDate>2012-06-15</InceptionDate>
      <StartOfFiscalYear>
        <Day>1</Day>
        <Month>1</Month>
      </StartOfFiscalYear>
      <EndOfFiscalYear>
        <Day>31</Day>
        <Month>12</Month>
      </EndOfFiscalYear>
      <OpenClosedEnded>OPEN</OpenClosedEnded>
      <Custodian>
        <Identifiers>
          <LEI>RCNB21CWB8HYAFVEL56</LEI>
        </Identifiers>
        <Name>Banque Internationale à Luxembourg S.A.</Name>
      </Custodian>
      <InvestmentCompany>
        <Identifiers>
          <LEI>549300ABCDEFGHIIJ1234</LEI>
        </Identifiers>
        <Name>Europa Asset Management S.A.</Name>
      </InvestmentCompany>
      <SFDRProductType>8</SFDRProductType>
    </FundStaticData>
  </Fund>
</Funds>

```

```

<FundDynamicData>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
      <TotalAssetNature>OFFICIAL</TotalAssetNature>
      <TotalNetAssetValue>
        <Amount ccy="EUR">464552848.78</Amount>
      </TotalNetAssetValue>
    </TotalAssetValue>
  </TotalAssetValues>
  <Portfolios>
    <Portfolio>
      <NavDate>2026-03-31</NavDate>
      <Positions>
        <Position>
          <UniqueID>ASSET-EQ-001</UniqueID>
          <Identifiers>
            <ISIN>DE0007164600</ISIN>
          </Identifiers>
          <Currency>EUR</Currency>
          <TotalValue>
            <Amount ccy="EUR">52340000.00</Amount>
          </TotalValue>
          <Equity>
            <Units>250000</Units>
            <Price>
              <Amount ccy="EUR">209.36</Amount>
            </Price>
          </Equity>
        </Position>
        <Position>
          <UniqueID>ASSET-B0-001</UniqueID>
          <Identifiers>
            <ISIN>DE0001102481</ISIN>
          </Identifiers>
          <Currency>EUR</Currency>
          <TotalValue>
            <Amount ccy="EUR">30125000.00</Amount>
          </TotalValue>
          <Bond>
            <Nominal>300000000</Nominal>
            <Price>
              <Amount ccy="EUR">100.4167</Amount>
            </Price>
          </Bond>
        </Position>
        <Position>
          <UniqueID>ASSET-AC-001</UniqueID>
          <Currency>EUR</Currency>
          <TotalValue>
            <Amount ccy="EUR">8750000.00</Amount>
          </TotalValue>
        </Position>
      </Positions>
    </Portfolio>
  </Portfolios>
</FundDynamicData>

```

```

        <Account>
            <MarketValue>
                <Amount ccy="EUR">8750000.00</Amount>
            </MarketValue>
        </Account>
    </Position>
    <Position>
        <UniqueID>ASSET-EQ-002</UniqueID>
        <Identifiers>
            <ISIN>FR0000120271</ISIN>
        </Identifiers>
        <Currency>EUR</Currency>
        <TotalValue>
            <Amount ccy="EUR">41875000.00</Amount>
        </TotalValue>
        <Equity>
            <Units>125000</Units>
            <Price>
                <Amount ccy="EUR">335.00</Amount>
            </Price>
        </Equity>
    </Position>
</Positions>
</Portfolio>
</Portfolios>
</FundDynamicData>
</Fund>
</Funds>
<AssetMasterData>
    <Asset>
        <UniqueID>ASSET-EQ-001</UniqueID>
        <Identifiers>
            <ISIN>DE0007164600</ISIN>
        </Identifiers>
        <Currency>EUR</Currency>
        <Country>DE</Country>
        <Name>SAP SE</Name>
        <AssetType>EQ</AssetType>
    </Asset>
    <Asset>
        <UniqueID>ASSET-B0-001</UniqueID>
        <Identifiers>
            <ISIN>DE0001102481</ISIN>
        </Identifiers>
        <Currency>EUR</Currency>
        <Country>DE</Country>
        <Name>Bundesrepublik Deutschland 0.00% 2026-02-15</Name>
        <AssetType>B0</AssetType>
    </Asset>
    <Asset>
        <UniqueID>ASSET-AC-001</UniqueID>
        <Currency>EUR</Currency>
    </Asset>

```

```

    <Country>LU</Country>
    <Name>BIL Custody Account EUR</Name>
    <AssetType>AC</AssetType>
  </Asset>
  <Asset>
    <UniqueID>ASSET-EQ-002</UniqueID>
    <Identifiers>
      <ISIN>FR0000120271</ISIN>
    </Identifiers>
    <Currency>EUR</Currency>
    <Country>FR</Country>
    <Name>TotalEnergies SE</Name>
    <AssetType>EQ</AssetType>
  </Asset>
</AssetMasterData>
</FundsXML4>

```

D.3 Full Production Delivery

A realistic monthly production delivery. This document exercises:

- **ControlData** with Contact details and two CustomAttributes (delivery sequence number and generator version).
- **FundStaticData** with Administrator, Auditor, Custodian, InvestmentCompany, OngoingCosts (Ongoing Charges and Transaction Costs), and SFDRProductType.
- **FundDynamicData** with a portfolio of eight positions: four equities (including one CHF-denominated position with dual-currency TotalValue), two bonds, one FX forward (CHF/EUR hedge), and one custody account.
- **Transactions** with a BUY and a SELL, each carrying the required NominalOrUnitsOrContracts.
- **SingleFund/ShareClasses** with three share classes (I-EUR-ACC, R-EUR-DIS, R-CHF-ACC-HEDGED).
- **AssetMasterData** with eight assets matching the portfolio positions (AssetType codes EQ, BO, FX, AC).
- **Documents** with a Prospectus reference at fund level and a PRIIPS-KID reference at share-class level.

```

<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
  <ControlData>
    <UniqueDocumentID>EGF-20260331-003</UniqueDocumentID>
    <DocumentGenerated>2026-04-01T08:30:00Z</DocumentGenerated>
    <Version>4.2.8</Version>
    <ContentDate>2026-03-31</ContentDate>
    <DataSupplier>
      <SystemCountry>LU</SystemCountry>
      <Short>EAM</Short>
      <Name>Europa Asset Management S.A.</Name>
      <Type>IC</Type>
    </DataSupplier>
  </ControlData>

```



```

    <Contact>
      <Name>Fund Data Operations</Name>
      <Email>fundsxml@eam-lu.example.com</Email>
    </Contact>
  </DataSupplier>
  <DataOperation>INITIAL</DataOperation>
  <Language>en</Language>
  <CustomAttributes>
    <Attribute>
      <Name>eam.delivery.sequenceNumber</Name>
      <Type>N</Type>
      <ValueNumber>42</ValueNumber>
    </Attribute>
    <Attribute>
      <Name>eam.delivery.generatorVersion</Name>
      <Type>T</Type>
      <ValueText>EAM-Pipeline 2.1.0</ValueText>
    </Attribute>
  </CustomAttributes>
</ControlData>
<Funds>
  <Fund>
    <Identifiers>
      <LEI>549300ABCDEFGHIIJ1234</LEI>
    </Identifiers>
    <Names>
      <OfficialName>Europa Growth Fund</OfficialName>
      <ShortName>EGF</ShortName>
    </Names>
    <Currency>EUR</Currency>
    <SingleFundFlag>true</SingleFundFlag>
    <FundStaticData>
      <DomicileCountry>LU</DomicileCountry>
      <ListedLegalStructure>UCITS</ListedLegalStructure>
      <InceptionDate>2012-06-15</InceptionDate>
      <StartOfFiscalYear>
        <Day>1</Day>
        <Month>1</Month>
      </StartOfFiscalYear>
      <EndOfFiscalYear>
        <Day>31</Day>
        <Month>12</Month>
      </EndOfFiscalYear>
      <OpenClosedEnded>OPEN</OpenClosedEnded>
      <Administrator>
        <Identifiers>
          <LEI>529900HNOAA1KXQJUQ27</LEI>
        </Identifiers>
        <Name>European Fund Administration S.A.</Name>
      </Administrator>
      <Auditor>
        <Identifiers>

```

```

        <LEI>1RVNBN7QG3CEEKYN6T68</LEI>
      </Identifiers>
      <Name>PricewaterhouseCoopers Société coopérative</Name>
    </Auditor>
    <Custodian>
      <Identifiers>
        <LEI>RCNB21CWB8H8HYAFVEL56</LEI>
      </Identifiers>
      <Name>Banque Internationale à Luxembourg S.A.</Name>
    </Custodian>
    <InvestmentCompany>
      <Identifiers>
        <LEI>5493000ABCDEFGHIJ1234</LEI>
      </Identifiers>
      <Name>Europa Asset Management S.A.</Name>
    </InvestmentCompany>
    <OngoingCosts>
      <OngoingCost>
        <CostType>Ongoing Charges</CostType>
        <PublicationDate>2025-12-31</PublicationDate>
        <Percentage>1.45</Percentage>
      </OngoingCost>
      <OngoingCost>
        <CostType>Transaction Costs</CostType>
        <PublicationDate>2025-12-31</PublicationDate>
        <Percentage>0.12</Percentage>
      </OngoingCost>
    </OngoingCosts>
    <SFDRProductType>8</SFDRProductType>
  </FundStaticData>
  <FundDynamicData>
    <TotalAssetValues>
      <TotalAssetValue>
        <NavDate>2026-03-31</NavDate>
        <TotalAssetNature>OFFICIAL</TotalAssetNature>
        <TotalNetAssetValue>
          <Amount ccy="EUR">464552848.78</Amount>
        </TotalNetAssetValue>
      </TotalAssetValue>
    </TotalAssetValues>
    <Portfolios>
      <Portfolio>
        <NavDate>2026-03-31</NavDate>
        <Positions>
          <Position>
            <UniqueID>ASSET-EQ-001</UniqueID>
            <Identifiers><ISIN>DE0007164600</ISIN></Identifiers>
            <Currency>EUR</Currency>
            <TotalValue><Amount ccy="EUR">52340000.00</Amount></TotalValue>
            <Equity>
              <Units>250000</Units>
              <Price><Amount ccy="EUR">209.36</Amount></Price>
            </Equity>
          </Position>
        </Positions>
      </Portfolio>
    </Portfolios>
  </FundDynamicData>
</Fund>

```

```

    </Equity>
  </Position>
  <Position>
    <UniqueID>ASSET-EQ-002</UniqueID>
    <Identifiers><ISIN>FR0000120271</ISIN></Identifiers>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">41875000.00</Amount></TotalValue>
    <Equity>
      <Units>125000</Units>
      <Price><Amount ccy="EUR">335.00</Amount></Price>
    </Equity>
  </Position>
  <Position>
    <UniqueID>ASSET-EQ-003</UniqueID>
    <Identifiers><ISIN>NL0010273215</ISIN></Identifiers>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">89600000.00</Amount></TotalValue>
    <Equity>
      <Units>100000</Units>
      <Price><Amount ccy="EUR">896.00</Amount></Price>
    </Equity>
  </Position>
  <Position>
    <UniqueID>ASSET-EQ-004</UniqueID>
    <Identifiers><ISIN>CH0012221716</ISIN></Identifiers>
    <Currency>CHF</Currency>
    <TotalValue>
      <Amount ccy="EUR">74250000.00</Amount>
      <Amount ccy="CHF">71280000.00</Amount>
    </TotalValue>
    <Equity>
      <Units>300000</Units>
      <Price><Amount ccy="CHF">237.60</Amount></Price>
    </Equity>
  </Position>
  <Position>
    <UniqueID>ASSET-B0-001</UniqueID>
    <Identifiers><ISIN>DE0001102481</ISIN></Identifiers>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">30125000.00</Amount></TotalValue>
    <Bond>
      <Nominal>30000000</Nominal>
      <Price><Amount ccy="EUR">100.4167</Amount></Price>
    </Bond>
  </Position>
  <Position>
    <UniqueID>ASSET-B0-002</UniqueID>
    <Identifiers><ISIN>FR0014007L00</ISIN></Identifiers>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">50312848.78</Amount></TotalValue>
    <Bond>
      <Nominal>50000000</Nominal>

```

```

        <Price><Amount ccy="EUR">100.6257</Amount></Price>
    </Bond>
</Position>
<Position>
    <UniqueID>ASSET-FX-001</UniqueID>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">-125000.00</Amount></TotalValue>
    <FXForward>
        <FxRateForEvaluation>1.0416</FxRateForEvaluation>
    </FXForward>
</Position>
<Position>
    <UniqueID>ASSET-AC-001</UniqueID>
    <Currency>EUR</Currency>
    <TotalValue><Amount ccy="EUR">126175000.00</Amount></TotalValue>
    <Account>
        <MarketValue>
            <Amount ccy="EUR">126175000.00</Amount>
        </MarketValue>
    </Account>
</Position>
</Positions>
<Transactions>
    <Transaction>
        <TransactionID>TXN-20260328-001</TransactionID>
        <AssetUniqueID>ASSET-EQ-003</AssetUniqueID>
        <Identifiers><ISIN>NL0010273215</ISIN></Identifiers>
        <Currency>EUR</Currency>
        <TransactionKind>BUY</TransactionKind>
        <ClosingDate>2026-03-28</ClosingDate>
        <NominalOrUnitsOrContracts>10000</NominalOrUnitsOrContracts>
    </Transaction>
    <Transaction>
        <TransactionID>TXN-20260330-001</TransactionID>
        <AssetUniqueID>ASSET-EQ-001</AssetUniqueID>
        <Identifiers><ISIN>DE0007164600</ISIN></Identifiers>
        <Currency>EUR</Currency>
        <TransactionKind>SELL</TransactionKind>
        <ClosingDate>2026-03-30</ClosingDate>
        <NominalOrUnitsOrContracts>5000</NominalOrUnitsOrContracts>
    </Transaction>
</Transactions>
</Portfolio>
</Portfolios>
</FundDynamicData>
<SingleFund>
    <ShareClasses>
        <ShareClass>
            <Identifiers><ISIN>LU1234567890</ISIN></Identifiers>
            <Names>
                <OfficialName>Europa Growth Fund I EUR Acc</OfficialName>
            </Names>

```

```

        <Currency>EUR</Currency>
    </ShareClass>
    <ShareClass>
        <Identifiers><ISIN>LU1234567891</ISIN></Identifiers>
        <Names>
            <OfficialName>Europa Growth Fund R EUR Dis</OfficialName>
        </Names>
        <Currency>EUR</Currency>
    </ShareClass>
    <ShareClass>
        <Identifiers><ISIN>LU1234567892</ISIN></Identifiers>
        <Names>
            <OfficialName>Europa Growth Fund R CHF Acc Hedged</OfficialName>
        </Names>
        <Currency>CHF</Currency>
    </ShareClass>
</ShareClasses>
</SingleFund>
</Fund>
</Funds>
<AssetMasterData>
    <Asset>
        <UniqueID>ASSET-EQ-001</UniqueID>
        <Identifiers><ISIN>DE0007164600</ISIN></Identifiers>
        <Currency>EUR</Currency>
        <Country>DE</Country>
        <Name>SAP SE</Name>
        <AssetType>EQ</AssetType>
    </Asset>
    <Asset>
        <UniqueID>ASSET-EQ-002</UniqueID>
        <Identifiers><ISIN>FR0000120271</ISIN></Identifiers>
        <Currency>EUR</Currency>
        <Country>FR</Country>
        <Name>TotalEnergies SE</Name>
        <AssetType>EQ</AssetType>
    </Asset>
    <Asset>
        <UniqueID>ASSET-EQ-003</UniqueID>
        <Identifiers><ISIN>NL0010273215</ISIN></Identifiers>
        <Currency>EUR</Currency>
        <Country>NL</Country>
        <Name>ASML Holding N.V.</Name>
        <AssetType>EQ</AssetType>
    </Asset>
    <Asset>
        <UniqueID>ASSET-EQ-004</UniqueID>
        <Identifiers><ISIN>CH0012221716</ISIN></Identifiers>
        <Currency>CHF</Currency>
        <Country>CH</Country>
        <Name>ABB Ltd</Name>
        <AssetType>EQ</AssetType>
    </Asset>

```

```

</Asset>
<Asset>
  <UniqueID>ASSET-B0-001</UniqueID>
  <Identifiers><ISIN>DE0001102481</ISIN></Identifiers>
  <Currency>EUR</Currency>
  <Country>DE</Country>
  <Name>Bundesrepublik Deutschland 0.00% 2026-02-15</Name>
  <AssetType>B0</AssetType>
</Asset>
<Asset>
  <UniqueID>ASSET-B0-002</UniqueID>
  <Identifiers><ISIN>FR0014007L00</ISIN></Identifiers>
  <Currency>EUR</Currency>
  <Country>FR</Country>
  <Name>République Française 0.00% 2027-02-25</Name>
  <AssetType>B0</AssetType>
</Asset>
<Asset>
  <UniqueID>ASSET-FX-001</UniqueID>
  <Currency>EUR</Currency>
  <Country>LU</Country>
  <Name>FX Forward CHF/EUR 2026-04-30</Name>
  <AssetType>FX</AssetType>
</Asset>
<Asset>
  <UniqueID>ASSET-AC-001</UniqueID>
  <Currency>EUR</Currency>
  <Country>LU</Country>
  <Name>BIL Custody Account EUR</Name>
  <AssetType>AC</AssetType>
</Asset>
</AssetMasterData>
<Documents>
  <Document>
    <Type><ListedType>Prospectus</ListedType></Type>
    <Language>en</Language>
    <Fund>
      <Identifiers><LEI>549300ABCDEFGHIIJ1234</LEI></Identifiers>
      <Name>Europa Growth Fund</Name>
    </Fund>
    <Format>PDF</Format>
    <CreationDate>2026-01-15</CreationDate>
    <DocumentURL>https://www.eam-lu.example.com/docs/egf-prospectus-
2026.pdf</DocumentURL>
  </Document>
  <Document>
    <Type><ListedType>PRIIPS-KID</ListedType></Type>
    <Language>en</Language>
    <ShareClasses>
      <ShareClass>
        <Identifiers><ISIN>LU1234567890</ISIN></Identifiers>
        <Name>Europa Growth Fund I EUR Acc</Name>

```

```

        </ShareClass>
    </ShareClasses>
    <Format>PDF</Format>
    <CreationDate>2026-02-01</CreationDate>
    <DocumentURL>https://www.eam-lu.example.com/docs/egf-kid-i-eur-acc-
2026.pdf</DocumentURL>
    </Document>
</Documents>
</FundsXML4>

```

D.4 AMEND Delivery

A correction delivery that supersedes D.2. The `DataOperation` is set to `AMEND`, and the `RelatedDocumentIDs` element references the original document's `UniqueDocumentID`. The corrected content — a revised TNAV — replaces the earlier value. Consumers that implement the three-operation protocol from Chapter 4 will locate the original delivery by its ID and apply the correction.

```

<?xml version="1.0" encoding="UTF-8"?>
<FundsXML4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="FundsXML4.xsd">
    <ControlData>
        <UniqueDocumentID>EGF-20260331-004</UniqueDocumentID>
        <DocumentGenerated>2026-04-02T09:15:00Z</DocumentGenerated>
        <Version>4.2.8</Version>
        <ContentDate>2026-03-31</ContentDate>
        <DataSupplier>
            <SystemCountry>LU</SystemCountry>
            <Short>EAM</Short>
            <Name>Europa Asset Management S.A.</Name>
            <Type>IC</Type>
        </DataSupplier>
        <DataOperation>AMEND</DataOperation>
        <RelatedDocumentIDs>
            <RelatedDocumentID>EGF-20260331-002</RelatedDocumentID>
        </RelatedDocumentIDs>
        <Language>en</Language>
    </ControlData>
    <Funds>
        <Fund>
            <Identifiers>
                <LEI>549300ABCDEFGHJIJ1234</LEI>
            </Identifiers>
            <Names>
                <OfficialName>Europa Growth Fund</OfficialName>
            </Names>
            <Currency>EUR</Currency>

```

```
<SingleFundFlag>true</SingleFundFlag>
<FundDynamicData>
  <TotalAssetValues>
    <TotalAssetValue>
      <NavDate>2026-03-31</NavDate>
      <TotalAssetNature>OFFICIAL</TotalAssetNature>
      <TotalNetAssetValue>
        <Amount ccy="EUR">464589123.45</Amount>
      </TotalNetAssetValue>
    </TotalAssetValue>
  </TotalAssetValues>
</FundDynamicData>
</Fund>
</Funds>
</FundsXML4>
```

D.5 Validation Instructions

To validate any of the files in this appendix, place the file and the FundsXML schema (FundsXML4.xsd , along with its dependency `xmlsig-core-schema.xsd`) in the same directory and run:

```
xmllint --schema FundsXML4.xsd filename.xml --noout
```

The expected output for each file is:

```
filename.xml validates
```

If validation fails, check that the `xsi:noNamespaceSchemaLocation` attribute points to the correct schema file relative to the XML file's location, and that both schema files are present.

The files in this appendix were validated against FundsXML schema version 4.2.8 at the time of writing. Newer schema versions may introduce additional required elements or tighten existing constraints; if a file that validated against 4.2.8 fails against a later version, consult the release changelog for the specific change that caused the failure.

Resources and Links

Official sources, GitHub repository, community, and training offerings

This appendix collects the external resources that a FundsXML practitioner is most likely to need. It is organised by category rather than alphabetically, so that a reader looking for a specific kind of resource can find the right section quickly. URLs are current as of early 2026; if a link has moved by the time you read this, the organisation name and resource description should give you enough context to find the new location.

E.1 The FundsXML Association

The FundsXML association is the non-profit body that stewards the standard. Its website is the authoritative source for governance information, membership details, working-group schedules, and announcements.

- **FundsXML association website** — <https://www.fundxml.org> — The official home of the standard. Contains the current schema files, the release changelog, the membership list, the board composition, and the working-group calendar. The website also hosts the association's contact form for enquiries about membership, schema questions, and working-group participation.
 - **FundsXML GitHub repository** — <https://github.com/fundxml> — The public repository where the schema files, sample documents, and related tooling are maintained. The issue tracker on the repository is the primary channel for reporting schema bugs, requesting clarifications, and proposing new features. Readers who want to contribute should start here. → Chapter 14 for the contribution process.
 - **FundsXML schema releases** — <https://github.com/fundxml/schema/releases> — Tagged releases of the FundsXML schema, each with the XSD file, changelog, and release notes. The URL pattern `https://github.com/fundxml/schema/releases/download/{version}/FundsXML.xsd` gives direct access to any version (see §4.7.2).
 - **FundsXML implementation guide** — <https://www.fundxml.org/implementation-guide/> — Practical guidance for getting started with the standard, including the root-element declaration and schema referencing.
 - **FundsXML downloads** — <https://www.fundxml.org/downloads/> — Schema files, sample documents, and supplementary materials available for direct download.
 - **FundsXML community** — <https://www.fundxml.org/community/> — Community resources, discussion channels, and announcements.
-

E.2 Schema and Specification Files

- **FundsXML4.xsd** — Available from <https://github.com/fundxml/schema/releases> and from the association website. The file is approximately 1.7 MB and defines the complete structure of a valid FundsXML 4.x document.
- **xmldsig-core-schema.xsd** — The W3C XML Digital Signature schema, imported by FundsXML4.xsd for the optional `ds:Signature` element. Available from <https://www.w3.org/TR/xmldsig-core/> and typically bundled with the FundsXML schema distribution.
- **FundsXML Online Schema Documentation** — <https://fundxml.github.io> — An interactive, browser-based reference for the FundsXML 4.x schema. Every element is navigable by XPath: appending `?xpath=/FundsXML4/ControlData` to the URL, for example, opens the documentation for the `ControlData` node directly. The documentation shows each element's type, cardinality, child elements, and annotations. This is the quickest way to look up a specific node without opening the XSD in an editor. Throughout this book, element names in the running text link directly to the corresponding page in this documentation. Table E.1 below lists the most important entry points.
- **FundsXML Release Changelog** — Published alongside each release on <https://github.com/fundxml/schema/releases>, the changelog enumerates every field added, removed, or changed. Essential reading when upgrading a pipeline from one version to the next.
- **FundsXML Documentation** — <https://www.fundxml.org/downloads/> — The association publishes supplementary documentation for the schema, including element-by-element descriptions, usage guidelines, and worked examples.

E.3 FinDatEx Templates

The five FinDatEx regulatory templates embedded in FundsXML are published and maintained by FinDatEx, not by the FundsXML association. Each template has its own specification document, field-by-field definition, and version history.

- **FinDatEx website** — <https://findatex.eu> — The official home of the European Working Group on Investment Data Exchange. Contains the current versions of all templates, the consultation documents for upcoming versions, and the working-group meeting calendar.
- **EMT (European MiFID Template)** — Current version: v4.3. The specification document defines approximately 150 fields covering target market, costs, risk classification, and distribution strategy. Published as an Excel workbook with field definitions and as a PDF specification.
- **EPT (European PRIIPs Template)** — Current version: v2.1. Defines the inputs needed for PRIIPs KID generation: performance scenarios, cost structures, risk indicators, and recommended holding periods.
- **EET (European ESG Template)** — Current version: v1.1.3. Carries SFDR-mandated sustainability disclosures: PAI indicators, Taxonomy alignment percentages, sustainable investment share, and exclusions policy.
- **EFT (European Feedback Template)** — Current version: v1.0. Facilitates the exchange of feeder-fund-specific information between feeder and master funds.
- **TPT (Tripartite Template)** — Current version: v7.0. Used for institutional Solvency II reporting of portfolio composition between asset managers, insurance clients, and their regulators.

E.4 Tooling

E.4.1 FreeXmlToolkit

FreeXmlToolkit is the open-source desktop application covered in Chapter 11. It provides an integrated environment for editing, validating, transforming, and signing FundsXML files.

- **FreeXmlToolkit GitHub repository** — <https://github.com/karlkauc/FreeXmlToolkit> — Source code, binary releases, issue tracker, and contribution guide. The application is written in Java and JavaFX and runs on Windows, macOS, and Linux.

E.4.2 Validation Tools

- **xmllint** — <https://gitlab.gnome.org/GNOME/libxml2> — The command-line XML validator from the libxml2 project. Available on virtually all Linux and macOS systems and installable on Windows via MSYS2 or WSL. Used throughout this book for schema validation. Invocation: `xmllint --schema FundsXML4.xsd file.xml --noout`.
- **Saxon** — <https://www.saxonica.com> — The reference XSLT 2.0 / 3.0 and XPath 2.0 / 3.1 processor, available in open-source (Saxon-HE) and commercial (Saxon-PE, Saxon-EE) editions. Useful for Schematron validation (which requires an XSLT processor) and for complex XPath queries.
- **lxml (Python)** — <https://lxml.de> — The Python binding for libxml2 and libxslt. Provides schema validation, XPath queries, XSLT transformation, and ISO Schematron support through the `lxml.isoschematron` module. Used in the Chapter 10 and Chapter 12 code examples.

E.4.3 Programming Libraries

The four languages covered in Chapter 12 each have standard XML libraries suitable for FundsXML work:

- **Java: JAXP** — The Java API for XML Processing, included in the standard JDK. Provides DOM, SAX, StAX parsing, XPath evaluation, schema validation, and XSLT transformation.
 - **Python: lxml** — See above. The `lxml.etree` module is the recommended library for FundsXML work in Python.
 - **C#: System.Xml.Linq (LINQ to XML)** — The .NET standard library for XML processing. Provides XDocument/XElement for document construction and query, plus XmlSchemaSet for validation.
 - **JavaScript/Node.js: fast-xml-parser** — <https://www.npmjs.com/package/fast-xml-parser> — A high-performance XML parser for Node.js. Used in the Chapter 12 example. Alternative: `xml2js` for a more DOM-like API.
-

E.5 Regulatory References

The regulatory frameworks that drive FundsXML's content — particularly the FinDatEx template content — are published by the European Commission and by ESMA. The primary references are:

- **UCITS Directive** — Directive 2009/65/EC of the European Parliament and of the Council. The foundational directive for European retail investment funds. Consolidated text available on EUR-Lex.
 - **AIFMD** — Directive 2011/61/EU on alternative investment fund managers. The companion directive for non-UCITS funds. AIFMD II (adopted 2024) amends the original.
 - **MiFID II** — Directive 2014/65/EU on markets in financial instruments. The directive that drives the EMT template's target-market and cost-disclosure requirements.
 - **PRIIPs Regulation** — Regulation (EU) No 1286/2014 on key information documents for packaged retail investment products. The regulation that drives the EPT template's KID-generation inputs.
 - **SFDR** — Regulation (EU) 2019/2088 on sustainability-related disclosures in the financial services sector. The regulation that drives the EET template's ESG disclosures.
 - **Taxonomy Regulation** — Regulation (EU) 2020/852 on the establishment of a framework to facilitate sustainable investment. Defines the criteria for environmentally sustainable economic activities referenced by the EET.
 - **ESAP Regulation** — Regulation (EU) 2023/2859 establishing a European Single Access Point. Scheduled to become operational in 2027; relevant for producers whose deliveries will flow into the centralised platform. → Chapter 14.
 - **EUR-Lex** — <https://eur-lex.europa.eu> — The official portal for EU law. All regulations and directives referenced above are available in all EU languages, with consolidated texts that incorporate subsequent amendments.
-

E.6 Standards and Identifier Registries

- **ISO 6166 (ISIN)** — The international standard for securities identification numbers. ISINs are issued by national numbering agencies and are the primary instrument identifier in FundsXML.
- **ISO 17442 (LEI)** — The international standard for legal entity identifiers. LEIs are issued by local operating units under the oversight of GLEIF.
- **GLEIF (Global LEI Foundation)** — <https://www.gleif.org> — The foundation that oversees the global LEI system. Its website provides a free LEI lookup service, bulk data downloads, and documentation on LEI validation (including the check-digit algorithm).
- **ISO 4217 (Currency Codes)** — The international standard for three-letter currency codes used in FundsXML's `ISOCurrencyCodeType`.
- **ISO 3166 (Country Codes)** — The international standard for two-letter country codes used in FundsXML's `ISOCountryCodeType`.
- **ISO 10962 (CFI)** — The Classification of Financial Instruments standard. Six-character codes that classify instruments by category, group, and attribute.
- **ISO 10383 (MIC)** — The Market Identifier Code standard. Four-character codes identifying trading venues.

E.7 Community and Learning

- **FundsXML annual conference** — The association organises an annual conference (typically in Vienna or a rotating European city) where practitioners, vendors, and regulators meet to discuss the state of the standard, share implementation experiences, and plan future development. The conference is open to members and, on a registration basis, to non-members.
 - **Working-group meetings** — The association's working groups meet monthly by video conference and are the primary venue for hands-on schema development. Participation is open to members and, on invitation, to external experts. → Chapter 14 for how to join.
 - **FundsXML on LinkedIn** — <https://www.linkedin.com/company/fundxml> — The association maintains a LinkedIn presence where announcements, conference invitations, and community news are posted.
 - **FundsXML on YouTube** — https://www.youtube.com/channel/UCzSMxta13eDIYQPEmH2_iww — Webinars, conference recordings, and tutorial videos.
 - **W3C XML specifications** — <https://www.w3.org/XML/> — The W3C website hosts the authoritative specifications for XML 1.0, XML Schema (XSD), XPath, XSLT, and XML Digital Signatures. Essential reading for practitioners who want to understand the technology layer beneath FundsXML. → Appendix B for a condensed reference.
 - **This book** — *FundsXML — The European Standard for Fund Data Exchange* is itself intended as a community resource. Feedback, corrections, and suggestions for future editions are welcome through the publisher's contact channels.
-

E.8 Companion Files from This Book

The following files accompany this book and are available for download from the book's repository. They are designed to be used together: the sample FundsXML file provides the input, and the XSLT stylesheets and programming-language scripts process it. Readers are encouraged to download all files, run the examples, and adapt them to their own FundsXML data.

E.8.1 Sample Data

File	Description	Chapters
europa_growth_fund.xml	Complete, schema-valid FundsXML 4.2.2 document for the fictional Europa Growth Fund. Contains ControlData, fund static data (domicile, entities), three share classes with prices and NAVs, fifteen portfolio positions (equities, bonds, FX forward, cash), and asset master data for all referenced instruments. This is the single input file for all examples below.	4–8, 12
FundsXML.xsd	The FundsXML 4.2.2 schema definition file, for offline validation. Validate the sample file with: <code>xmllint --noout --schema FundsXML.xsd europa_growth_fund.xml</code>	2, 10

E.8.2 XSLT Stylesheets

All stylesheets are XSLT 1.0 and can be run with `xsltproc` (bundled with libxml2 on Linux and macOS), with Saxon, or with any XSLT-capable library (Python `lxml`, Java `javax.xml.transform`, .NET `System.Xml.Xsl`).

File	Description	Chapter
examples/xslt/shareclass_csv.v.xsl	Extracts a multi-column CSV of share-class data (fund name, LEI, ISIN, share-class name, currency, NAV date). Run: <code>xsltproc examples/xslt/shareclass_csv.xsl europa_growth_fund.xml</code>	§2.12.2
examples/xslt/factsheet_simple.xsl	Produces a self-contained HTML fact sheet with fund metadata table (LEI, currency, NAV, dates) and a share-class listing.	§2.12.3
examples/xslt/dq_nav_check.xsl	Data-quality check: verifies that portfolio position values sum to the fund's total net asset value, with rounding tolerance.	§2.12.4
examples/xslt/factsheet.xml	Produces a self-contained HTML fact sheet with a table showing LEI, currency, NAV, NAV date, and content date.	§12.3.7
examples/xslt/positions_csv.xsl	Exports all portfolio positions as a CSV file, joining position data with asset master data via the UniqueID/IDREF mechanism.	§12.3.7

E.8.3 Programming-Language Examples

Each script reads or processes `europa_growth_fund.xml`. Run instructions and dependency notes are included as comments at the top of each file.

Python (requires `lxml` ; install with `pip install lxml`)

File	Description	Chapter
<code>examples/python/read_fund.py</code>	Reads a FundsXML file and prints the fund name, LEI, and total NAV.	§12.3.2
<code>examples/python/import_fund.py</code>	Parses a FundsXML file and imports fund data and portfolio positions into PostgreSQL. Requires <code>psycopg2</code> .	§12.5.1
<code>examples/python/export_fund.py</code>	Queries fund data from PostgreSQL and reconstructs a FundsXML fragment. Requires <code>psycopg2</code> .	§12.5.1

Java (JDK 17+; external libraries noted per file)

File	Description	Chapter
<code>examples/java/ReadFund.java</code>	Reads a FundsXML file with JAXP and XPath (no external dependencies).	§12.3.3
<code>examples/java/FundsXmlToMongo.java</code>	Imports a FundsXML file into MongoDB. Requires <code>mongodb-driver-sync</code> .	§12.5.2
<code>examples/java/MongoToFundsXml.java</code>	Exports fund data from MongoDB and builds a FundsXML fragment with JAXP DOM. Requires <code>mongodb-driver-sync</code> .	§12.5.2
<code>examples/java/FundsXmlToBaseX.java</code>	Imports a FundsXML file into a BaseX XML database. Requires <code>org.basex:basex</code> .	§12.5.3
<code>examples/java/BaseXQuery.java</code>	Queries and exports fund data from BaseX using embedded XQuery. Requires <code>org.basex:basex</code> .	§12.5.3

C# (.NET 8+)

File	Description	Chapter
<code>examples/csharp/ReadFund.cs</code>	Reads a FundsXML file with LINQ to XML (no external dependencies).	§12.3.4

JavaScript / Node.js (requires `fast-xml-parser`; install with `npm install fast-xml-parser`)

File	Description	Chapter
<code>examples/javascript/read_fund.mjs</code>	Reads a FundsXML file with <code>fast-xml-parser</code> .	§12.3.5

E.9 FundsXML Online Schema Documentation — Quick Reference

The interactive schema documentation at <https://fundsxml.github.io> allows direct navigation to any element via an XPath parameter in the URL. The general pattern is:

```
https://fundsxml.github.io/index.html?xpath=/FundsXML4/...
```

Table E.1 lists the most frequently needed entry points. Each URL opens the documentation page for the corresponding node, showing its type definition, cardinality, child elements, and schema annotations.

Table E.1 — Key entry points into the FundsXML Online Schema Documentation

Schema Area	XPath	URL	Treated in
Root element	/FundsXML4	fundxml.github.io/...?xpath=/FundsXML4	Chapter 3
ControlData	/FundsXML4/ControlData	fundxml.github.io/...?xpath=/FundsXML4/ControlData	Chapter 4
Funds container	/FundsXML4/Funds	fundxml.github.io/...?xpath=/FundsXML4/Funds	Chapter 5
Fund element	/FundsXML4/Funds/Fund	fundxml.github.io/...?xpath=/FundsXML4/Funds/Fund	Chapter 5
FundStaticData	/FundsXML4/Funds/Fund/FundStaticData	fundxml.github.io/...?xpath=/FundsXML4/Funds/Fund/FundStaticData	Chapter 5
FundDynamicData	/FundsXML4/Funds/Fund/FundDynamicData	fundxml.github.io/...?xpath=/FundsXML4/Funds/Fund/FundDynamicData	Chapters 5, 6
Share Classes	/FundsXML4/Funds/Fund/SingleFund/ShareClasses	fundxml.github.io/...?xpath=/FundsXML4/Funds/Fund/SingleFund/ShareClasses	Chapter 5
Portfolio	/FundsXML4/Funds/Fund/FundDynamicData/Portfolios	fundxml.github.io/...?xpath=/FundsXML4/.../Portfolio	Chapter 6

Schema Area	XPath	URL	Treated in
	/Portfolio		
Positions	/FundsXML4/Funds/FundDynamicData/Portfolios/Portfolio/Positions	fundsxml.github.io/...?xpath=/FundsXML4/.../Positions	Chapter 6
AssetMasterData	/FundsXML4/AssetMasterData	fundsxml.github.io/...?xpath=/FundsXML4/AssetMasterData	Chapter 6
Transactions (Flows)	/FundsXML4/Funds/Fund/SingleFund/ShareClasses/ShareClasses/Flows	fundsxml.github.io/...?xpath=/FundsXML4/.../Flows	Chapter 7
RegulatoryReportings	/FundsXML4/RegulatoryReportings	fundsxml.github.io/...?xpath=/FundsXML4/RegulatoryReportings	Chapter 8
Documents	/FundsXML4/Documents	fundsxml.github.io/...?xpath=/FundsXML4/Documents	Chapter 9
CountrySpecificData	/FundsXML4/CountrySpecificData	fundsxml.github.io/...?xpath=/FundsXML4/CountrySpecificData	Chapter 9

Schema Area	XPath	URL	Treated in
	ecific-Data		

Readers are encouraged to bookmark the root URL and explore the schema tree interactively. The documentation is generated directly from the XSD and therefore always reflects the structure of the published schema version.

INDEX

Index

Page references point to the chapter and section where the topic is treated substantively, not to every passing mention. Bold entries indicate the primary treatment. Appendix references use the letter prefix (A, B, C, D, E). The Glossary (Appendix A) is not indexed separately; readers looking for a definition should consult it directly.

A

Acceptance testing, **13.6.1**
Account (AssetType AC), 6, C.4.3, C.4.4
Accumulating share class, 5, A.1
AIFMD (Alternative Investment Fund Managers Directive), **1**, 8, A.1, E.5
AMEND (DataOperation), **4**, 10, 13.5.3, C.7.3, D.4
AmountType, **C.7.1**
API facades, **14.3.2**
Asset management company, *see* Investment company
AssetMasterData, **6**, C.4.4, D.2, D.3
AssetType enumeration (two-character codes), **C.4.4**
Audit log, 12.2.1, **13.7.1**
Auditor (FundStaticData), C.3.3

B

Base currency, 5, A.1
Batch delivery, **1**, **12.2**, 14.3.1, A.1
Benchmarks (FundStaticData/FundDynamicData), C.3.3, C.3.4
Bond position, 6, C.4.2, C.4.3, D.2
Business rules, *see* Schematron

C

Cash account position, 6, C.4.2, C.4.3, D.2
CDATA sections, B.1.3
CFI (Classification of Financial Instruments), A.1, E.6
Character escaping (XML), **B.1.3**

Classifications (FundStaticData), C.3.3
Comdirect Bank (pilot consumer), 13.4.4
Comments (XML), B.1.4
Community contributions, **14.5**
CompanyType, C.3.3
Compliance sign-off, 13.6.1
Consumer pipeline, **12.2.2**, A.1
ContentDate, **4**, C.2.1
ControlData, **4**, C.2, D.1
Corporate action (CORP_ACTION), 7, C.5.1
CountrySpecificData, **9**, C.8.2
Cross-border distribution, 1, 9, 14.2.2, A.1
Crypto (AssetType CR), C.4.4
Currency codes (ISO 4217), 5, E.6
CUSIP, A.1, C.7.2
CustomAttributes, **9**, C.8.1, D.3
Custodian, 1, 5, C.3.3, A.1
Cutoff time, A.1
Cutover versus parallel run, **13.6.2**, 13.9

D

Dashboard (operations), 13.7.1
Data gaps, **13.3.3**
Data quality issues, **13.3.4**
DataOperation (INITIAL/AMEND/DELETE), **4**, C.2.1, C.7.3
DataSupplier, **4**, C.2.2
DataSupplierType, C.2.2
DELETE (DataOperation), **4**, 10, C.7.3
Digital assets (tokenised funds), 14.2.2
DirectReporting, **C.6.1**
Dispatcher, **12.2.3**, A.1
Distribution country, 1, 9, A.1
DocumentGenerated, **4**, C.2.1
Documents block, **9**, C.7.5, D.3
DocumentType (listed types), C.7.5
Domicile, 5, C.3.3, A.1
DOM parsing, 12.3

E

Earnings, 7, C.4.1, C.5.2

EET (European ESG Template), **8**, C.6.1, E.3

EFT (European Feedback Template), **8**, C.6.1, E.3

Element order (XML schema), 2, 10, C

EMT (European MiFID Template), **8**, C.6.1, E.3

Emission window, 13.7.1, A.1

Enumerations, *see* individual enumeration names; summary **C.7.3**

EPT (European PRIIPs Template), **8**, C.6.2, E.3

Equity position, 6, C.4.2, C.4.3, D.2

ESAP (European Single Access Point), **14.3.3**, E.5

ESG, *see* SFDR; EET

ESMA, 1, 8, A.1, E.5

ETF (Exchange-Traded Fund), A.1

EUR-Lex, E.5

Europa Asset Management S.A., *passim*; implementation project **13**

Europa Growth Fund, *passim*; complete examples **D**

Event-driven architecture, 14.3.4, A.1

Extension points, *see* CustomAttributes; CountrySpecificData

F

Fair value, A.1

fast-xml-parser (Node.js), **12.3**, E.4.3

Feeder fund, 8, A.1

FIGI, A.1, C.7.2

FinDatEx, **8**, 14.2.4, E.3

Fixture (test fixture), **13.5.2**, A.1

FreeXmlToolkit, **11**, 14.5.5, E.4.1

Fund (FundType), **5**, C.3.1

Fund administrator, 1, A.1

FundAmountType, C.7.1

FundDynamicData, **5**, C.3.4

FundStaticData, **5**, C.3.3

FundsXML association, 1, **14.2.3**, E.1

FundsXML4 (root element), **C.1**

FX forward position, C.4.3, D.3

G

Generator (FreeXmlToolkit), 11, A.1
GitHub (FundsXML repository), 14.5.2, E.1
GLEIF, A.1, E.6
Go-live, *see* Acceptance testing; Parallel run
Golden fixture, 13.5.2, A.1
Governance, **14.2.3**
GraphQL facade, 14.3.2

H

Hedged share class, 5, A.1

I

Idempotency, 12.2.2, A.1
Identifiers (IdentifiersType), **C.7.2**
Implementation project, **13**
IndirectReporting, **C.6.2**
INITIAL (DataOperation), **4**, C.7.3
Integration test, 13.5.1, A.1
InvestmentCompany (FundStaticData), C.3.3
ISIN, 5, 6, C.7.2, E.6

J

Java (JAXP), **12.3**, E.4.3
JSON versus XML, **14.3.1**

K

KID (Key Information Document), 8, A.1
Knowledge transfer, **13.7.1**, A.1

L

Language (ControlData), 4, C.2.1

Legal structure, *see* ListedLegalStructure

LEI (Legal Entity Identifier), 4, 5, C.7.2, E.6

ListedLegalStructure enumeration, **C.7.4**

Locale-dependent number formatting (bug), 13.5.3, 13.6.3

lxml (Python), **10**, **12.3**, E.4.2

M

Mailing list (FundsXML), 14.5.3, E.1

Mapping (data mapping), **13.3**

Mapping table, **13.3.2**, Table 13.1

Master fund, A.1

MiFID II, **8**, E.5

ModuleUsage, C.2.3

Monthly delivery (example), **D.2**

N

Namespace, **4**, B.1.5, A.1

NAV (Net Asset Value), **5**, A.1

NamesType, C.3.2

Node.js, *see* fast-xml-parser

NominalOrUnitsOrContracts (Transaction), C.5.1

Non-functional requirements, 13.2.2

O

OFFICIAL (TotalAssetNature), **5**, C.7.3

OngoingCosts (FundStaticData), C.3.3, D.3

OpenClosedEnded enumeration, C.7.3

Operations and maintenance, **13.7**

P

PAI (Principal Adverse Impacts), **8**, A.1

Parallel run, **13.6.2**, 13.9, A.1

Paying agent, A.1

Performance (pipeline), 13.6.3

Pipeline, *see* Producer pipeline; Consumer pipeline

Portfolio (PortfolioType), **6**, C.4.1

Position (PositionType), **6**, C.4.2

Position ordering, 13.4.3

PRIIPs, **8**, E.5

Producer pipeline, **12.2.1**, A.1

Prospectus, A.1, D.3

Prototype, **13.4**, A.1

Python, *see* lxml

R

Redemption, 7, A.1

RegulatoryReportings, **8**, C.6

RelatedDocumentIDs, **4**, 10, C.2.4, D.4

REST facade, **14.3.2**

Runbook, **13.7.1**, A.1

S

SAX parsing, 12.3

Saxon (XSLT processor), E.4.2

Schema (FundsXML4.xsd), **C**, E.2

Schema upgrades, **13.7.3**, 14.2.1

Schematron, **10**, B.2, E.4.2

SEDOL, A.1, C.7.2

SFDR (Sustainable Finance Disclosure Regulation), **8**, 14.2.2, E.5

SFDRProductType enumeration, C.3.3, C.7.3

Share class (ShareClassType), **5**, C.3.7, D.3

Signature (XML Digital Signature), **9**, C.1

SingleFund / Subfunds (choice), **5**, C.3.1, C.3.6

SingleFundFlag, 5, C.3.1

Source system, 13.2.2, 13.3, A.1

StAX parsing, 12.3

Streaming architecture, 14.3.4

Sub-fund, 5, A.1

Subscription, 7, A.1

Swing pricing, A.1

T

T+1 / T+2 settlement, 14.2.2, A.1

Target market, 8, A.1

Taxonomy Regulation, 8, E.5

TECHNICAL (TotalAssetNature), 5, C.7.3

Test fixtures, *see* Fixture

Testing (implementation project), **13.5**

Text types (Text16Type through Text500Type), C.7.6

TNAV (Total Net Asset Value), 5, C.3.5, A.1

TotalAssetNature enumeration, **C.7.3**

TotalAssetValueType, **C.3.5**

TPT (Tripartite Template), **8**, C.6.2, E.3

Transaction (TransactionType), 7, C.5.1, D.3

TransactionKind enumeration, **C.5.1**, C.7.3

Transfer agent, A.1

Two-stage validation, **10**, 12.2.1

U

UAT (User Acceptance Testing), **13.6.1**, A.1

UCITS, **1**, E.5

Umbrella fund, 5, C.3.1, A.1

UniqueDocumentID, 4, C.2.1

Unit test, 13.5.1, A.1

V

Validation (XSD + Schematron), **10**, D.5

Validation failures (error messages), 10

Valuation point, A.1

Version (ControlData), 4, C.2.1

W

Warehouse loader, **12.2.4**

WKN (Wertpapierkennnummer), A.1, C.7.2

Working groups (FundsXML association), 14.2.3, **14.5.4**, E.7

X

XML declaration, B.1.1

XML Digital Signature, *see* Signature

xmllint, **10**, D.5, E.4.2

XPath, **B.2**; FundsXML recipes B.2.6

XSD (XML Schema Definition), **2**, **10**, B, C

XSLT, **B.3**; FundsXML recipes B.3.6; FreeXmlToolkit 11