FDC3: What's New in 3.0?

Kris West **with Chris Watson**

NatWest Group

EW ELGIN WHITE CONSULTING

Agenda

- FDC3 recap & adoption
- The road to 3.0
- **Metadata & Observability**
- **Security, Identity & Zero-Trust Architectures**
- **Language Interoperability**
- **API Improvements**
- Getting involved & what's next
- **Demo** — FDC3 Sail 3.0

What is FDC3?

An open standard for **application interoperability** on the financial desktop, hosted by FINOS.

- Defines a shared language, based on **context** objects (shared data) & **intents** (actions), and a broker (the **Desktop Agent**) for communication
- In production at major banks, asset managers and vendors (io.Connect, Here and others)
- FDC3 2.2 introduced *FDC3 for Web Browsers* — enabling browser-native Desktop Agents & zero-install deployments

> *Apps that speak FDC3 work together without bespoke integrations.*

The Road to 3.0

FDC3 3.0 is focused on on four themes that **expand interoperability**:

1. **Observability** — where did context come from? What's the chain of events?
2. **Security** — how do you trust what you're receiving?
3. **Cross-Firm Interoperability** — competing apps, different organisations
4. **Cross-Technology Support** — backend services, multi-language, AI agents

Metadata & Observability

Understanding what happened

The Metadata Problem (FDC3 2.2)

In FDC3 2.2, when an app received context, it had no answers to:

- **When** was it created?
- **Which workflow or app** generated it?
- **Is it related** to a previous action?
- **Can I trust** the sender?

The *optional* `ContextMetadata` contained only one field:

```
interface ContextMetadata {
  readonly source: AppIdentifier;  // @experimental — who sent it
}
```

ContextMetadata in FDC3 3.0

In **FDC3 3.0** ContextMetadata is a **required** feature of the API and has been greatly expanded:

```
interface ContextMetadata {
  readonly source: AppIdentifier;          // Who sent it (now MUST)
  readonly timestamp: Date;                // When it was broadcast
  readonly traceId: string;                // Correlation ID for the workflow
  readonly signature?: DetachedSignature;  // Cryptographic proof of origin
  readonly antiReplay?: AntiReplayClaims;  // Replay prevention
  readonly custom?: Record<string, any>;   // Extensible — see next slide
}
```

Applications can **provide** metadata when broadcasting or raising intents:

```
interface AppProvidableContextMetadata {
  traceId?: string;              // Correlate a multi-app workflow
  signature?: DetachedSignature; // Sign the context (see Security)
  antiReplay?: AntiReplayClaims; // Prevent replay attacks
  custom?: Record<string, any>;  // Custom extensions
}
```

What Metadata Enables

Metadata, particularly `trace-id` and the `timestamp` have been often requested by FDC3 Stakeholders:

Stakeholder	Benefit
Developers	Trace context flow across apps for debugging
Architects	Visibility into inter-app communication patterns
Compliance	Provenance and audit trail for every exchange
Operations	Correlation IDs for distributed tracing

The `custom` Field — A Proving Ground

The `custom` field in `ContextMetadata` enables developers to explore capabilities not yet standardised:

- **Layout hints** when opening apps (position, size)
- **Priority or urgency** indicators
- **Vendor-specific routing metadata**

If viable, these can be incorporated into future drafts of FDC3.

Security & Identity

The headline feature of FDC3 3.0

(built on the API's expanded Metadata support)

Cross-Firm Workflows — Original Driver

We've shown cross-firm demos at past OSFF conferences. The need is clear:

Competing applications coexist on the same desktop.

- Two price providers — neither should see the other's data
- Requests must provably come from the right application (not a competitor)
- Data returned must come from the trusted source
- Open channels must not leak sensitive data to other apps

These requirements drove the design of FDC3 Security from the start.

Internal Zero-Trust — A Newer Use Case

Regulated firms increasingly require **provable data access**:

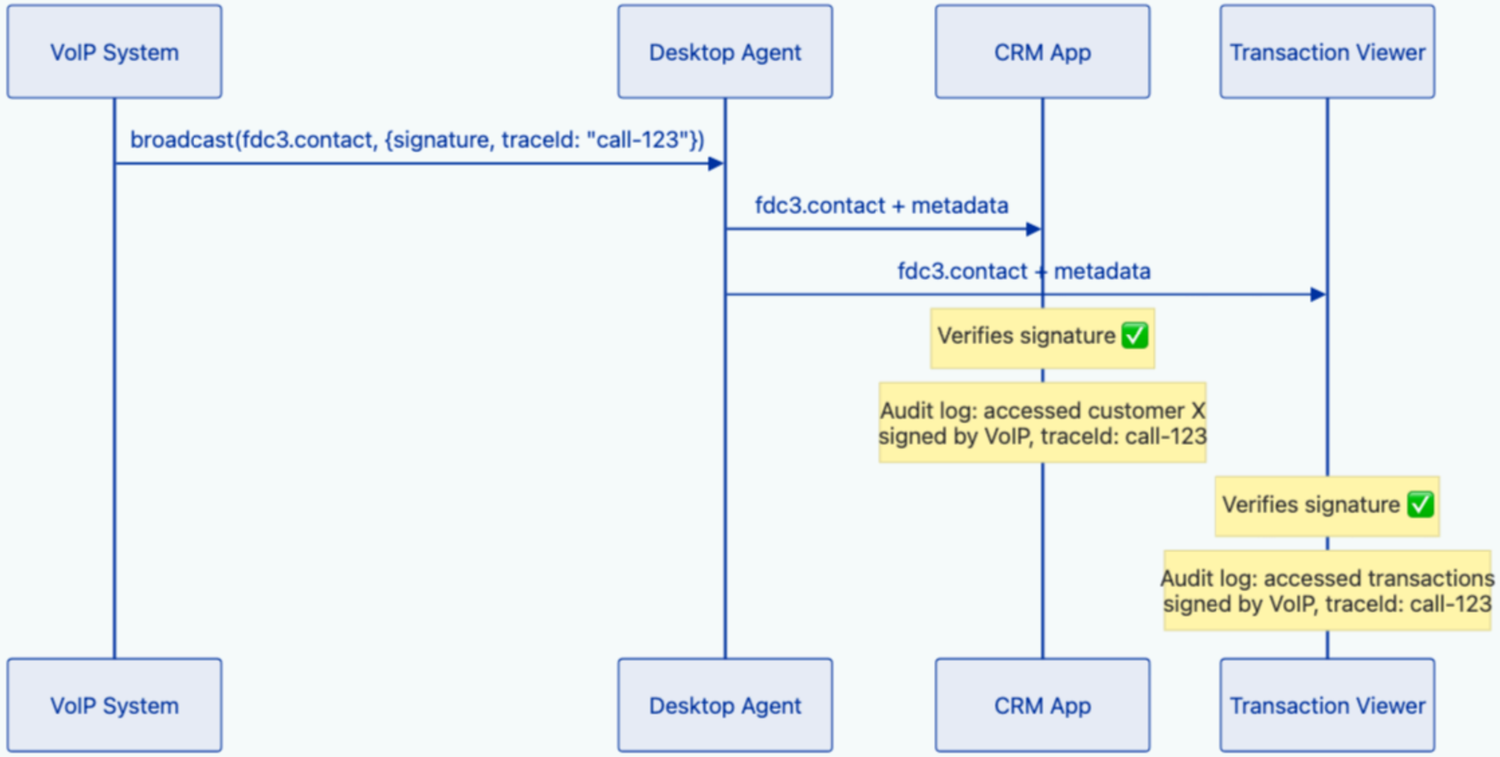
> *A customer calls in. The VoIP system broadcasts their details. A CRM, transaction viewer, and account system each retrieve further data.*

Auditors want to know: why was each app allowed to access this data?

- The VoIP or chat system **signs** the initial customer context
- Downstream apps can **prove** they received the identifier from an authorised source
- The `traceId` ties the interaction into one auditable workflow
- **Encrypted channels** ensure least privilege — only encrypted blobs in transit

Proving Data Access

The VoIP or chat system initiates a workflows and sets a `trace-id:`



Each app can prove — months later — it had a **legitimate, signed reason** to access the data & **telementry** can tie individual operations together

FDC3 3.0 Security Capabilities

Both **cross-firm integration** and **zero-trust architecture** need the same set of primitives:

Capability	What it enables
Signed contexts	Prove sender identity, detect tampering
Encrypted channels	Protect data from DA and observers
Symmetric key exchange	TLS-like efficient encrypted streams
User identity (JWT)	Portable authentication across apps
Anti-replay protection	Prevent replayed messages

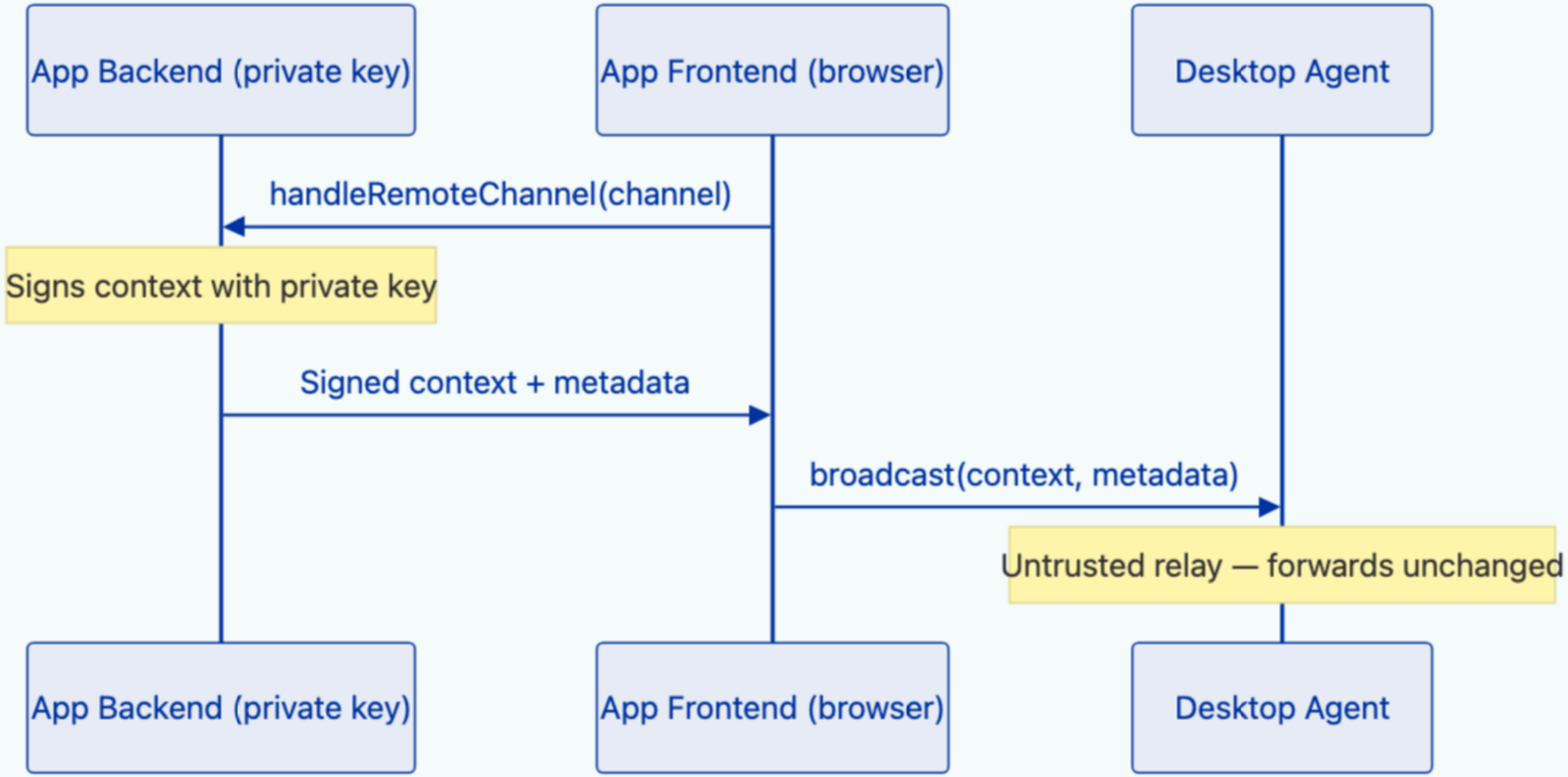
Trust Without Trusting the DA

The security model is built on **asymmetric cryptography**:

- Each app holds a **private key** (on a trusted backend — never in the browser)
- Each app publishes a **public key** at a stable HTTPS endpoint (JWKS - RFC 7517)
- Trust is determined by **applications**, not the Desktop Agent
- The DA is an **untrusted intermediary**

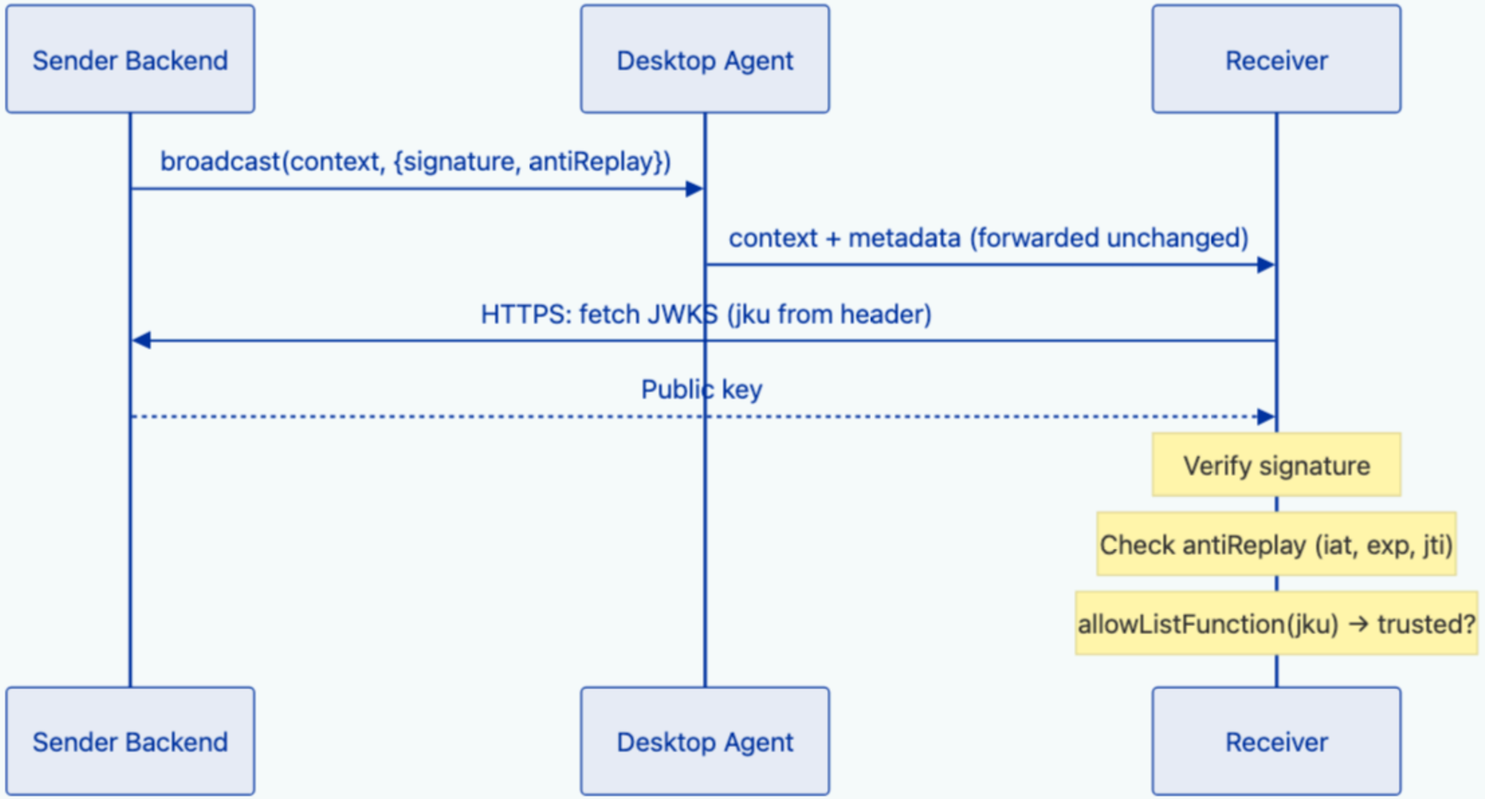
The Trusted Backend Contract

Cryptographic operations **MUST** run in a trusted backend — not in the browser.



Signing & Verification Flow

JSON Web Signatures (RFC 7515) are carried in the metadata fields (`signature` + `antiReplay`) and point to public key (jku):



Verification in Code

Signature verification is added to app's context handler:

```
// Allowlist: which signers does this app trust?
const TRUSTED = new Set([
  'https://voip-app.example.com/.well-known/jwks.json',
  'https://crm.example.com/.well-known/jwks.json'
]);
const allowList = (jku: string) => TRUSTED.has(jku);

// Context handler with verification
fdc3.addContextListener('fdc3.instrument', (context, metadata) => {
  const { authenticity } = securityImpl.verify(context, metadata, allowList);
  if (!authenticity?.signed) handleUntrustedContext(context);
  else if (!authenticity.valid) rejectContext(context);
  else if (!authenticity.trusted) promptUser(context, authenticity.jku);
  else processVerifiedInstrument(context); // ✅
});
```

Encrypted Communications

Signatures prove **who** sent a message — but don't prevent others **reading** it.

Encrypted channels use **symmetric AES-GCM**, where a symmetric key is exchanged via **JWE** (RFC 7516):

1. Broadcaster creates a symmetric key
2. Receiver requests the key (signed request)
3. Broadcaster wraps key with receiver's public key (JWE)
4. Stream encrypted with cheap symmetric cipher

> *Same model as TLS — asymmetric cost once, then efficient symmetric encryption.*

User Identity — The `GetUser` Intent

Building on signing and encryption, we can support **portable user identity**:

1. App raises `GetUser` intent with a **signed** request
2. Identity provider **verifies** the requesting app's signature
3. IDP mints a **signed, encrypted JWT** scoped to the requesting app — only the requestor can read it
4. Requester's backend decrypts and verifies the JWT

The user's identity is protected - but can be used to enable login & onboarding workflows

`fdc3-security` — Reference impl.

A new NPM package (`@finos/fdc3-security`) provides a **full reference implementation**:

- **Signed broadcasts** — `BasicSignedBroadcaster` + `SignatureChecking`
- **Encrypted channels** — `EncryptedBroadcastSupport` + `EncryptedContextListenerSupport`
- **Mutual intent authentication** — `SignedRaiseIntentSupport`
- **User identity** — `GetUser` intent with JWE-wrapped JWT
- **Trusted backend contract** — `FDC3Handlers` + WebSocket bridge
- **Anti-replay** — `DefaultAntiReplayChecker`

Apps can adopt the library directly — or implement the standard's procedures themselves.

Implementation Samples

Standalone samples

(mock DA, execute in console):

- Signed broadcasts
- Mutual intent authentication
- Encrypted channels (backend & frontend key)
- User identity (`GetUser`)

```
kristopher.west@M-CG6RXYGY02 fdc3-security % npx tsx samples/signing-broadcast-example.ts
=== FDC3 Signing Example Start ===
1. Starting App A backend...
2. Starting App B backend...
3. App A Setup: Exporting channel to backend (BasicSignedBroadcaster lives on backend)...
WebSocket client connected
[App A Backend] Received channel via handleRemoteChannel, wrapping with BasicSignedBroadca
ster
4. App B Setup: Wrapping channel handler (verification uses jku from signatures)...
[MockChannel] Listener added { channelId: 'fdc3.channel.1', listenerCount: 1 }

[App A Backend] Signing and broadcasting instrument...
[MockChannel] Broadcasting context {
  channelId: 'fdc3.channel.1',
  context: {
    type: 'fdc3.instrument',
    id: { ticker: 'AAPL' },
    name: 'Apple Inc.'
  },
  metadata: {
    signature: {
      protected: 'eyJhbGciOiJFZERTQSIsImprdSI6Imh0dHA6Ly8xMjcuMC4wLjE6NjQ4NTIvLndlbGwta25v
d24vandrcy5qc29uIiwiaWF0IjoxNzgyMTk3MTY1LCJraWQiOiJzaWduaW5nLTE3ODIxOTcxNjUxMzgtbmI5Z2VqNn
BzIn0',
      signature: 'MvAV01NeK3xXtc2_Ca8-9KnFDCpW6Q0pIHwRVZM5IhP6YFbrSEjkO3y_UE0tDR5yRGOw2-LX
-CTKaLQkpl8iDA'
    },
    antiReplay: {
      iat: 1782197165,
      exp: 1782197465,
      jti: 'd0176234-28c3-4013-b803-3e7c8401508f'
    }
  }
}
[App B] <<< [VERIFIED] Context Received:
{
  "type": "fdc3.instrument",
  "id": {
    "ticker": "AAPL"
  },
  "name": "Apple Inc."
}
[App B] <<< [VERIFIED] Metadata Received:
{
  "signature": {
    "protected": "eyJhbGciOiJFZERTQSIsImprdSI6Imh0dHA6Ly8xMjcuMC4wLjE6NjQ4NTIvLndlbGwta25v
d24vandrcy5qc29uIiwiaWF0IjoxNzgyMTk3MTY1LCJraWQiOiJzaWduaW5nLTE3ODIxOTcxNjUxMzgtbmI5Z2VqNn
BzIn0",
    "signature": "MvAV01NeK3xXtc2_Ca8-9KnFDCpW6Q0pIHwRVZM5IhP6YFbrSEjkO3y_UE0tDR5yRGOw2-LX
-CTKaLQkpl8iDA"
  },
  "antiReplay": {
```

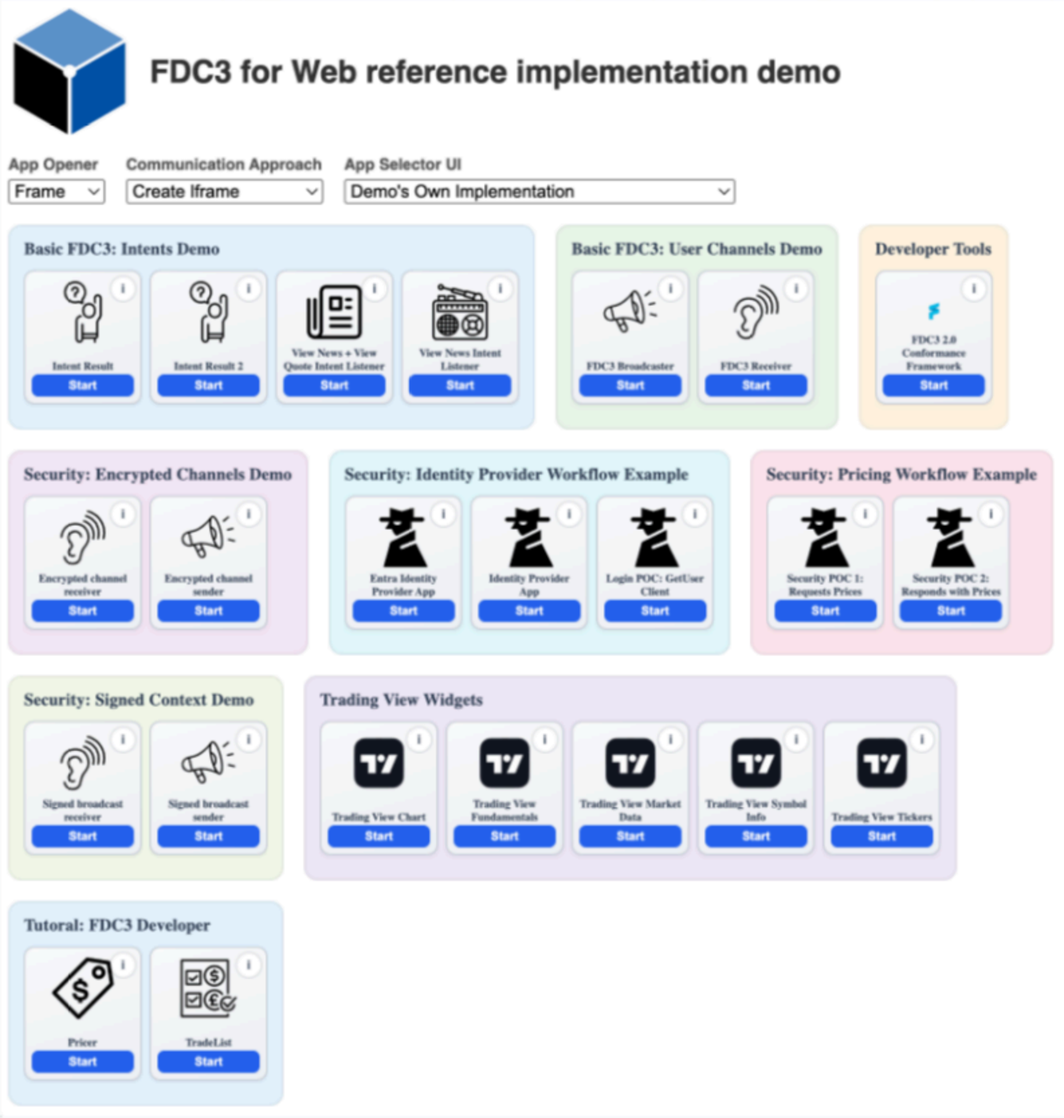
Example Applications

Full example apps

(Run them in the FDC3 for Web demo or your own DA):

- Signed Context Example
- Encrypted Channel Example
- Identity Provider Workflow
- Pricing Workflow Example

Based on OSFF 2025 demos



Language Interoperability

Beyond JavaScript

Why Multiple Languages?

FDC3 for **.NET** and **Java** have existed in vendor products for years:

- The fdc3-dotnet project provides a standard API binding & util classes
- But internal comms between vendor libs & DAs are **un-standardised**
  - Each vendor implements their own proprietary wire format
  - Apps are tied to a particular DA
  - Making it hard to build an ecosystem

The **Desktop Agent Communication Protocol (DACP)** (FDC3 2.2) + a proposed **WebSocket Connection Protocol (WSCP)** change this:

- Any language that can connect to a WebSocket can implement an FDC3 adaptor
- Far less work for DA vendors to support new languages

FDC3 3.0 Language Bindings

FDC3 3.0 will ship with official bindings for four languages:

Language	Package	Status
TypeScript / JavaScript	`@finos/fdc3-standard`	Updated
.NET / C#	`finos/fdc3-dotnet`	Updated
Go	In FDC3 repo	New in 3.0
Java	`finos-labs/fdc3-java-api`	New in 3.0

With **WSCP** + **DACP**, no vendor-specific integration or library is required.

API Improvements

The smaller (but useful) things

Channel: clearContext()

Applications can now **clear** context from a channel:

```
await channel.clearContext('fdc3.instrument'); // Specific type
await channel.clearContext();                  // All types
```

Listeners receive a `contextCleared` event. Useful when workflows end or stale data should be removed:

```
await channel.addEventListener("contextCleared", event => {
  console.log(`${event.details.type} context cleared on channel ${channel.id}`);
  //do something with the event such as clearing a filter
  resetFilter(event.details.type);
});
```

Other Notable API Changes

Feature	What it does
`fdc3.close()`	Apps can request their own window/frame be closed
`addIntentListenerWithContext()`	Filter intent listeners by context type
Intent listener conflicts	New error for rejecting overlapping listeners for the same intent
Metadata API functions	Retrieve current context *with* its metadata, return `ContextWithMetadata` as an intent result
Deprecated API removal	Functions deprecated in 2.0–2.2 removed (e.g. `getSystemChannels`, `joinChannel`, name-based `open`/`raiseIntent` etc.)
Destructuring support	All interface methods bound to instances
Fully-qualified appIds	Resolution procedures for cross-agent/multiple appD app identifiers
Request a new instance	Raise an Intent and guarantee a resolver won't pop up

Getting Involved

Try It Now

FDC3 3.0 is **not yet complete**, but you can start exploring today:

- **Alpha release** available on NPM: `@finos/fdc3@3.0.0-alpha.2`
- **Fully working reference implementation** for FDC3 for Web Browsers in the FDC3 repository
- Explore the new metadata, security and API features in your own apps

Get Involved

FDC3 is open to new contributors:

- Join working group meetings, mailing lists, and Slack
- **Contribute** to items in the **3.0 scope**
- Always looking for new **maintainers and editors** for the standard
- Look out for **good first issue** tags
- Maintainer onboarding

github.com/finos/FDC3#getting-involved

Demo: FDC3 Sail 3.0

Chris Watson

FDC3 Sail

FDC3 Sail is the FINOS community's production-ready Desktop Agent for the web.

- Built on the *FDC3 for Web Browsers* standard
- Implements FDC3 2.2 - working on FDC3 3.0 support
- Open source under the FINOS umbrella
- 3.0 is the first version designed to be **production-ready**

> *The FDC3 reference implementation (`fdc3-web-impl`) is being marked as **not for production use** from 3.0 — it exists to clarify the standard and test client-side code. **FDC3 Sail** will be the recommended path for production use.*

Demo

Contributors & Maintainers

FDC3 3.0 is the work of many hands. Special thanks to FDC3's maintainers and many other individual contributors from across the community:

Contributor	Organisation	Focus area
Rob Moffat	FINOS	FDC3 Security & Java Binding
Yannick Malins	Symphony	FDC3 Security
Vinay Mistry	Symphony	Use Cases
Lizaveta Kemereva	BlackRock Aladdin	FDC3 Go Binding & API
Giles Roadknight	Morgan Stanley	API & FDC3 for the Web
Brian Ingenito	Morgan Stanley	FDC3 .NET
Hugh Troeger	FactSet	FDC3 Editor
Johan Sandersson	Here	Governance
Juliana Mealin	interop.io	FDC3 Metadata
Kris West	NatWest Group	FDC3 Security, Metadata & Lead-Maintainer

Thank You

FDC3 3.0 Secure by design. Observable by default. Ready for the next decade.

- GitHub: github.com/finos/FDC3
- FDC3 Sail: github.com/finos/FDC3-Sail
- Docs: fdc3.finos.org

Kris West — kristopher.west@natwest.com

Chris Watson — christopher.watson@elginwhite.com