Building RESTful Web Services APIs

RESTful web services

- Representational State Transfer
  - An architectural style, not a product, transport, or protocol
  - A set of constraints used for creating web services
  - A method of accessing & manipulating textual representations of resources in a uniform and stateless way
- Architectural goals
  - Accessibility from any environment
  - Simplicity through use of a uniform interface
  - Performance and scalability
- RESTful implementations leverage various standards
  - URI RFC 3986
  - HTTP RFC 7230 - 7237
  - JSON RFC 7159

Why REST?

- Simple to learn, build, and use
  - URLs, HTTP, and basic CRUD operations
- Simple to write and document
- Completely open, reach more clients
- Less overhead
- Less duplication
- More standardized
- Testable
- Around 70% of public APIs are implemented as RESTful services

Harmony Core

- Our answer to the question of “How do I get there from here?”

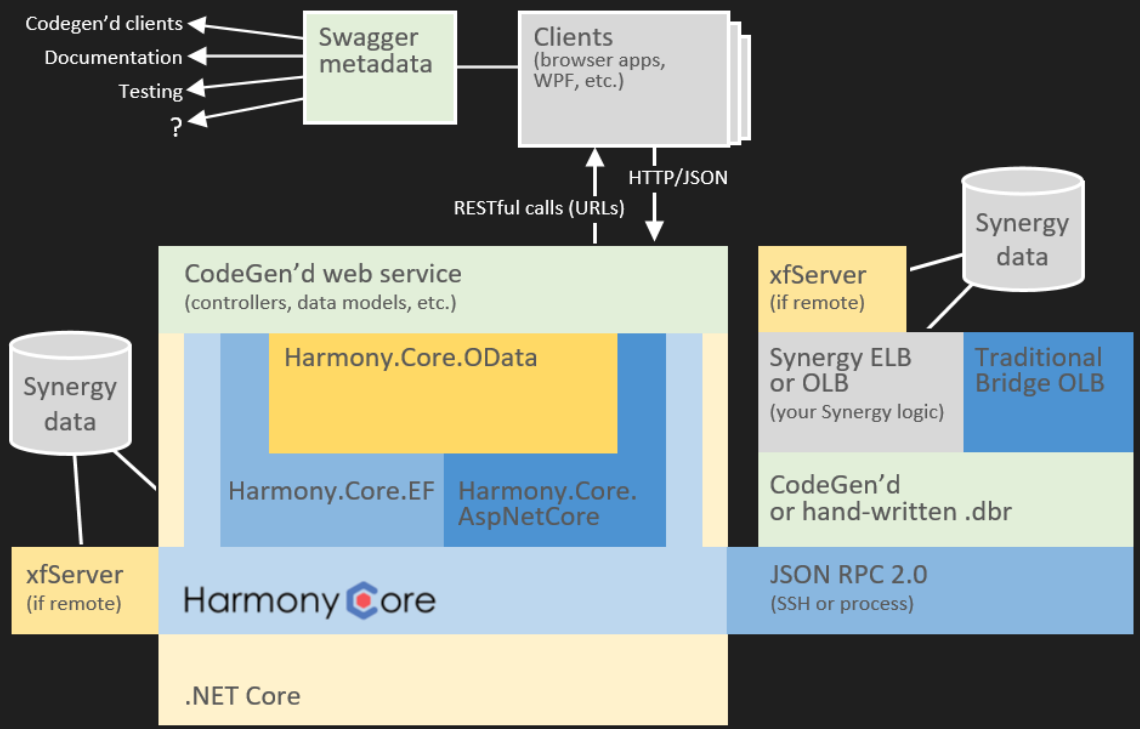
A framework of patterns, practices, CodeGen templates, and libraries used to create RESTful web services that expose Synergy data and logic

Why did we make Harmony Core?

- APIs make you sticky
- Integrating deeply with other software gives customers better experiences
- UIs change like the wind
- Single coherent answer to the question of “How?”
- Servicing model needed to allow us to evolve and adapt

What are the parts?

- EF Core provider
- OData
- Routing conventions
- Code isolation
  - In process
  - Out of process
  - On a separate machine
- CodeGen
- .NET Core



SYNERGEX

EF Core - read

- Translates LINQ expression trees into Synergy Select objects or READ/READS/FIND statements
- Translation is cached and not recomputed each time
- Uses Sparse functionality where appropriate
- Implementation reflects current best practices for data retrieval

EF Core – write

- Translates into STORE or WRITE statements by default
- Works with IOHooks
- Custom key generation for STORE
- Custom validation using partial methods
- Custom file I/O routines by extending FileIO classes
- Participates in transactions

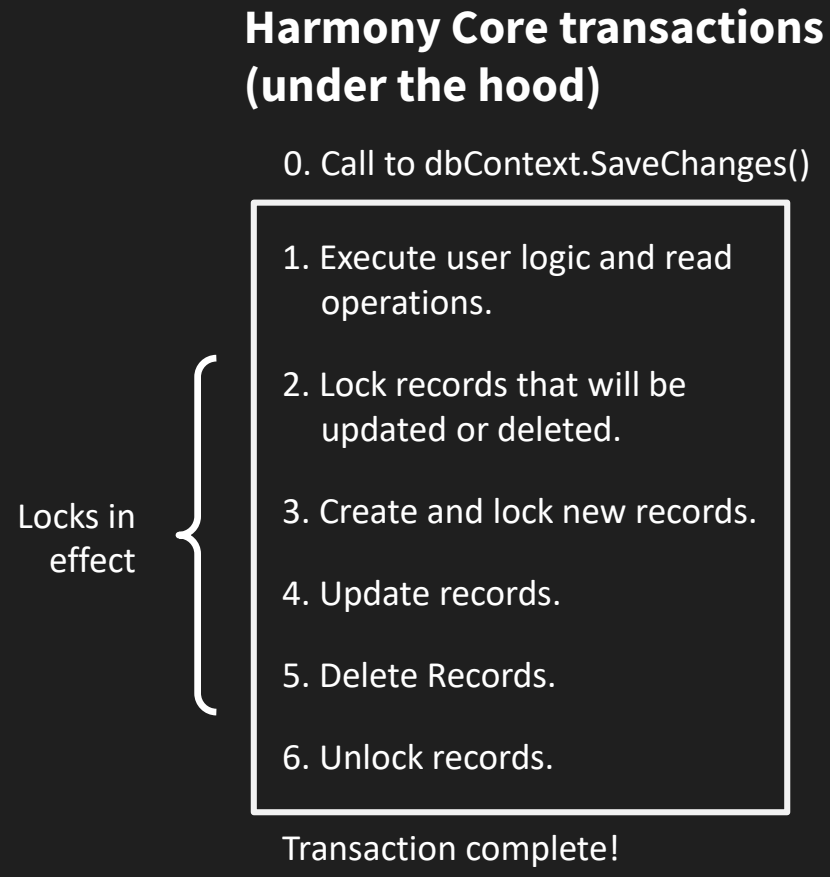
EF Core - patch

- Uses JsonPatchDocument standard
  - **Add:** add a value into an object or array.
  - **Remove:** remove a value from an object or array.
  - **Replace:** replace a value. Logically identical to using remove and then add.
  - **Copy:** copy a value from one path to another by adding the value at a specified location to another location.
  - **Move:** move a value from one place to another by removing from one location and adding to another.
  - **Test:** test for equality at a certain path for a certain value.

EF Core - transactions

- “Read committed” transaction isolation
- Optimistic concurrency based on GRFAs
- Nothing is written until SaveChanges is called
- SaveChanges tracks all objects produced by the DbContext
- All tracked objects with changes persist
- SaveChanges either succeeds or fails in its entirety
  - No leftovers after a failure

Sounds nice, but how?



SYNERGEX

But what happens if there's an error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

Locks in effect

1. Execute user logic and read operations.
2. **Lock records that will be updated or deleted.**
3. Create and lock new records.
4. Update records.
5. Delete Records.
6. Unlock records.

Points of failure

Unable to get a lock? (E.g., record already locked or deleted by another program)
Response: Throw an exception.

Different GRFA?
Response: Throw an exception.

Hardware failure?
Response: Throw an exception.

SYNERGEX

But what happens if there's an error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

Locks in effect

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. **Create and lock new records.**
4. Update records.
5. Delete Records.
6. Unlock records.

Points of failure

Unable to create a record? (E.g., duplicate key or disk full)
Response: Roll back created records in reverse order and throw an exception.

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction.)

But what happens if there's an error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. Create and lock new records.
4. **Update records.**
5. Delete Records.
6. Unlock records.

Locks in effect

Points of failure

Unable to update a record? (E.g., duplicate alternate key where duplicates are not allowed)
Response: Delete all created records (in reverse order), and write original values to any updated records.

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction)

SYNERGEX

But what happens if there's an error?

Harmony Core transactions (under the hood)

0. Call to dbContext.SaveChanges()

1. Execute user logic and read operations.
2. Lock records that will be updated or deleted.
3. Create and lock new records.
4. Update records.
5. **Delete Records.**
6. Unlock records.

Locks in effect

Points of failure

Hardware failure?
Response: Roll back everything that can be rolled back in reverse order, and throw an exception. (Incomplete transaction)

SYNERGEX

EF Core – materialization

- DataObject creation is customizable using partial methods
- Joined objects can have additional processing

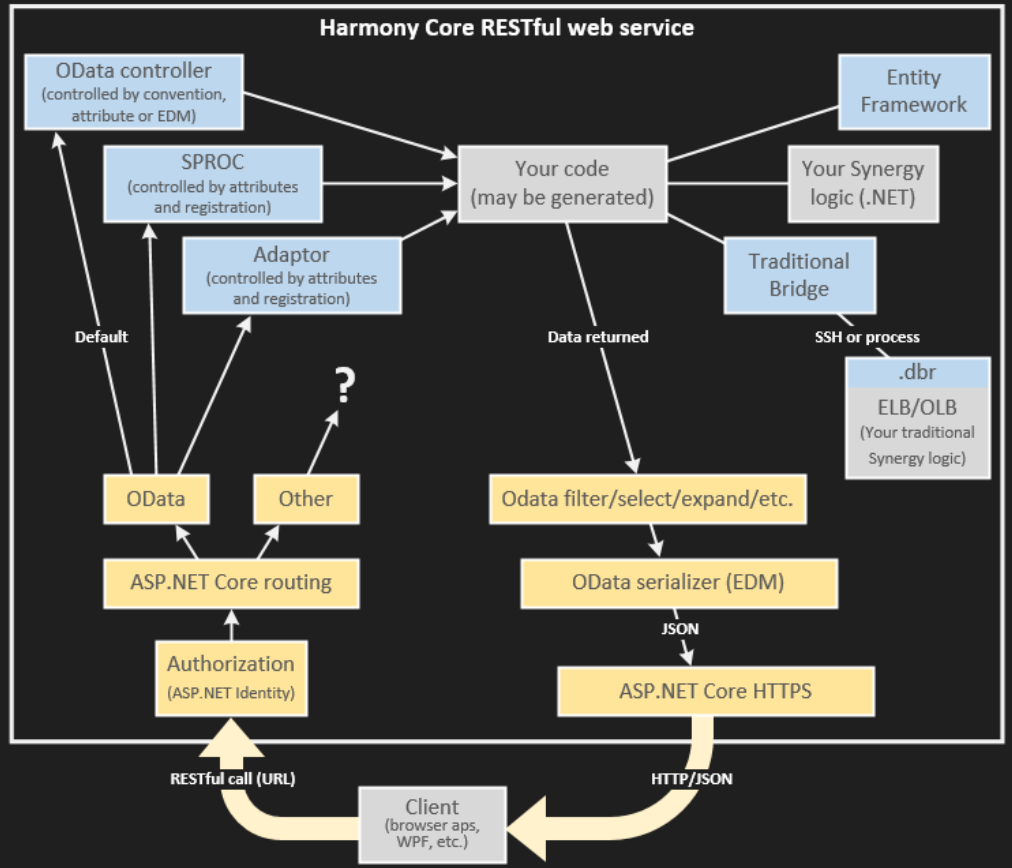
  E.g., turn multiple notes records into a single notes object

- Efficiently iterates the Join Select enumerator
  - Creates a sorted materialization plan once per query
  - Follows current best practices for unchanged result rows

EF Core – relations

- Understood by Harmony Core extensions for CodeGen
- Controlled by repository and JSON customization file
- Relation types
    A. (many) --> (one) --> (many)
    B. (one) --> (one) --> (one)
    C. (one) --> (one)
    D. (one) --> (many) --> (one)
    E. (one) --> (many)

URL routing



SYNERGEX

URL routing conventions

- Default OData
  - Controller name
  - GET/POST/PUT/PATCH/DELETE
  - OData $ operators like filter and expand
- ASP.NET Core MVC
  - Controller name
  - Attribute for HTTP method
  - Method name or attribute for name

URL routing conventions

- Sproc routing
  - Attribute on methods
  - Configurable security
  - Register the controller
  - Produces OData actions/functions
- Adapter routing
  - Similar to Sproc routing
  - Uses attributes on parameters to translate $filter conditions

Code isolation

- Globals, commons, statics
  - Cause problems when used with multiple threads
  - Operations that rely on that shared state behave incorrectly if state is changed by a different operation while they are running
  - Can be isolated using AssemblyLoadContext
    - Like AppDomains but much cheaper/faster (.NET Core only)
- Fixed channel numbers
  - Not currently isolatable using AssemblyLoadContext
  - Please don’t do this

Code isolation

- Run in a separate process
  - Standard in/out
  - JSON-RPC 2.0
  - Works with traditional Synergy
  - Can work with Synergy .NET as well
  - Can be made to work with other languages
- Run on a separate machine
  - SSH
  - Windows/Linux/Unix/OpenVMS

Traditional Bridge

- Why not *xf*ServerPlus?
  - *xf*ServerPlus uses a proprietary protocol
  - Pooling can be difficult
  - IT admins don't like things they don't understand
  - Security
    - Normal login and user privileges
    - Encrypted from start to finish
  - Data compression
- What if I already have *xf*ServerPlus?
  - CodeGen now understands method catalogs
  - We can generate wrappers for your *xf*ServerPlus code to use this instead
  - If it works for you, you can keep using it

Authentication & authorization

- ASP.NET Identity
  - Highly customizable
  - Supports multiple persistence mechanisms
    - ADFS, Azure AD, OData, etc.
  - Role provider supports custom access control
  - Claims-based authentication
  - Unit testable
- Consumer authenticates and obtains a “bearer token”
  - JSON Web Tokens (JWT)
- Bearer token presented back via authorization header
- Capabilities for protecting APIs
  - Authentication at the controller or operation level
  - Role-based authorization at the endpoint level
  - Role-based authorization at the field level

Authentication
Who are you?

Authorization
What can you do?

SYNERGEX

Multi-tenancy

- Different sites with different ISAM files
- Different sites with different *xf*Servers
- App.UseMultiTenancy(MethodUsedToDescriminateTenants)
  - The method you provide must take an @HttpContext and return a string
  - HttpContext gives you access to authentication, cookies and the HTTP headers
  - We recommend using the x-tenant-id HTTP header
- services.AddSingleton<**IFileChannelManager**, **YourFileChannelManager**>()
- Extend FileChannelManager or HookableFileChannelManager
  - Override GetChannel
  - Write code to translate requested filenames into tenant-specific file names
  - TenantId is available for the request using **MultiTenantProvider**.TenantId

Where we've been successful so far

- RESTful ODBC++
  - It's not really ODBC – but the use case is the same
  - Puts guardrails on the ways a client can query to prevent operations that might take hours to complete
  - May be read-only access, but some users might not have access to all data
  - HTTPS instead of TCP/IP
  - External consumers like the OData standard
- Custom code with AssemblyLoadContext isolation
  - Efficient way to deploy Synergy .NET code that uses globals/commons
  - OData actions, functions, and resources

For more information…

- https://github.com/Synergex/HarmonyCore - Source, issues, wiki
- https://synergex.github.io/HarmonyCore/ - Development blog
- https://www.synergex.com/web-services - Links to resources
- https://www.synergex.com/videos-harmony-core/ - Videos

- **Office Hours** – ask questions, get help
  - May 16th – 8:00AM PST
  - June 25th – 8:00AM PST
  - August 7th – 8:00AM PST

SYNERGEX

Who has the first question?