

BridgeControl

HTTP API Documentation

Initial API Specification and Agreed Conventions

API version: v1

Table Of Contents

1. Purpose and Scope	4
2. API Design Principles	4
3. Common API Conventions	5
4. Authentication and Session Management	6
4.1 Platform authentication	6
4.1.1 Get Current Platform Administrator	6
4.2 Vendor and Customer authentication	7
5. Platform Vendor Administration	9
6. User, Role and Access Administration	10
7. Customer Administration	11
8. Product Management	12
9. Product Version Management	13
10. MSI and JSON Schema Management	13
10.1 MSI installer	13
10.2 JSON Schema	13
11. Configuration Management	14
11.1 Product Default configuration	14
11.2 Customer configuration	14
11.3 Installation configuration	15
12. Effective Configuration	15
13. Configuration Migration	17
14. Installation Management	18
15. Licensing	18
16. Desired and Reported Installation State	20
16.1 Desired state	20
16.2 Reported state	20
17. Deployment Status and History	22
18. Agent Enrollment and Authentication	23
19. Agent Communication API	24
19.1 Heartbeat	24
19.2 Managed Installations	24
20. Agent Artifact Downloads	25
21. Agent Deployment Execution and Reporting	26
22. Logs and Monitoring	27
22.1 Agent log submission	27
22.2 Administrative monitoring	27
23. Authorization Model	28
24. Error and ProblemDetails Model	29
24.1 Validation example	29

24.2 Business conflict example	29
25. Standard HTTP Status Codes	30
26. Core Validation Rules	31
27. OpenAPI and Swagger	32
28. Persistence Considerations Identified by the API	33
29. Completion Criteria for the Initial API Design	34

1. Purpose and Scope

This document defines the initial HTTP API contract for BridgeControl. The API is used by the BridgeControl Web UI and the BridgeControl Agent and is implemented by the central ASP.NET Core Backend API.

The specification defines resources, routes, HTTP methods, DTOs, validation rules, authorization requirements, status codes, error responses, versioning, pagination, and OpenAPI conventions before controller or endpoint implementation.

Tenant model: A Software Vendor is the tenant. SOFTWARE_VENDOR.Id is the internal tenant identifier. SOFTWARE_VENDOR.Name is the user-facing Vendor identifier used during Vendor/Customer login. Vendor-scoped API routes do not accept a vendorId from the caller; the Backend resolves the Vendor from the authenticated identity.

2. API Design Principles

- The BridgeControl Backend API is the only component permitted to access the Platform DB, resolved Vendor DBs, and Vendor File Archives directly.
- The UI and Agents communicate with the Backend API over HTTPS.
- Platform administration is separated from Vendor-scoped administration.
- Vendor-specific resources are resolved from authentication context rather than a caller-supplied vendorId.
- Customer and Installation access is additionally constrained by ownership and assigned permissions.
- DTOs are API contracts and are not required to match EF Core entities one-to-one.
- Large files are streamed through controlled API endpoints while relational databases store file metadata and references.
- Desired installation state and Agent-reported state are separate resources.

3. Common API Conventions

Convention	Decision
Base path	/api/v1
Transport	HTTPS only
Default payload	application/json
JSON property naming	camelCase
Identifiers	Guid
Timestamps	DateTimeOffset; transmitted as UTC ISO 8601
Enums	String values
Error format	RFC Problem Details (application/problem+json)
Pagination	page and pageSize
Default page size	25
Maximum page size	100
File upload	multipart/form-data
Authentication	Bearer access token
Versioning	URL versioning: /api/v1/...
OpenAPI document	/openapi/v1.json
Swagger UI	/swagger

Paged collection responses use a common response envelope:

```
public sealed record PagedResponse<T>(  
    IReadOnlyList<T> Items,  
    int Page,  
    int PageSize,  
    int TotalCount);  
  
{  
    "items": [],  
    "page": 1,  
    "pageSize": 25,  
    "totalCount": 142  
}
```

Idempotency: GET is read-only. PUT represents replacement/set semantics and repeated identical requests must not create duplicate state. PATCH is used for partial updates. POST is used for commands and resource creation where repeated execution may produce a new operation.

4. Authentication and Session Management

4.1 Platform authentication

Method	Route	Purpose	Authorization
POST	/api/v1/platform/auth/login	Platform administrator login	Anonymous
POST	/api/v1/platform/auth/refresh	Renew platform session	Refresh credential
POST	/api/v1/platform/auth/logout	End platform session	Authenticated
GET	/api/v1/platform/account/me	Get current platform administrator	Platform administrator bearer token

4.1.1 Get Current Platform Administrator

Returns account details for the currently authenticated platform administrator. The account is derived from the validated platform access token.

GET /api/v1/platform/account/me

Authorization: Bearer <accessToken>

Requires HTTPS and an active platform administrator account and session. No request body, path parameters or query parameters are required. Tenant and Agent tokens are not accepted.

Successful response — 200 OK

```
{
  "id": "8cd52d63-5340-48d8-a789-b3175a8b004e",
  "email": "admin@example.com",
  "status": "Active",
  "createdAt": "2026-09-16T08:00:00+00:00",
  "lastLoginAt": "2026-09-17T09:30:00+00:00"
}
```

Response fields: id (UUID), email (string), status (string), createdAt (DateTimeOffset), and lastLoginAt (nullable DateTimeOffset). The last login timestamp is null when no login has been recorded.

Error responses

400 Bad Request: transport requirements are not met, such as using HTTP instead of HTTPS.

401 Unauthorized: the platform token is missing, invalid, expired or revoked, or the account/session is no longer valid.

403 Forbidden: the authenticated caller does not satisfy the platform administrator authorization policy.

500 Internal Server Error: an unexpected server failure occurred.

Errors use the standard Problem Details format. This operation is read-only; it does not renew sessions or issue tokens. Password hashes and infrastructure secrets are excluded. Responses must not be cached.

4.2 Vendor and Customer authentication

Method	Route	Purpose	Authorization
GET	/api/v1/auth/vendors	List active Vendor names	Anonymous
POST	/api/v1/auth/login	Vendor/Customer login	Anonymous
POST	/api/v1/auth/refresh	Renew session	Refresh credential
POST	/api/v1/auth/logout	End session	Authenticated
POST	/api/v1/auth/activate	Activate invited account	Activation token
POST	/api/v1/auth/password/forgot	Request password reset	Anonymous
POST	/api/v1/auth/password/reset	Reset forgotten password	Reset token
POST	/api/v1/auth/password/change	Change current password	Authenticated

Vendor discovery: Before login or password recovery, the UI retrieves active Vendor names from GET /api/v1/auth/vendors and presents a selector. The Backend queries SOFTWARE_VENDOR in the Platform DB and includes only records with Status = Active, ordered by Name using the database's case-insensitive collation.

Authorization: Anonymous; no bearer token or refresh credential is required. Each item contains only Name. Vendor IDs, user accounts, database names, connection references and storage details are not returned. Listing a Vendor does not authorize access; login still verifies Vendor readiness, lifecycle and user credentials. The full /api/v1/platform/vendors endpoint remains restricted to PlatformAdmin.

Pagination: page is one-based and defaults to 1; pageSize defaults to 25 and is limited to 1-100. The response is PagedResponse<LoginVendorResponse>. The UI retrieves all pages for the selector. An empty result returns 200 with items = [] and totalCount = 0.

```
public sealed record LoginVendorResponse(string Name);
```

Example: GET /api/v1/auth/vendors?page=1&pageSize=25 returns 200 OK:

```
{
  "items": [
    { "name": "Nordic Software" },
    { "name": "Northwind" }
  ],
  "page": 1,
  "pageSize": 25,
  "totalCount": 2
}
```

Transport and errors: HTTPS is required and responses use Cache-Control: no-store. Invalid pagination or transport/hosting checks return 400. Discovery is limited to 60 requests per minute per client IP, independently of login; excess requests return 429.

Because Vendor and Customer accounts are stored in isolated Vendor databases, the login request identifies the Software Vendor by name. The Backend resolves that Vendor in the Platform DB, obtains its internal Vendor Id, resolves the Vendor DB, and then verifies the user account.

```

public sealed record LoginRequest(
    string VendorName,
    string Email,
    string Password);

public sealed record AuthSessionResponse(
    Guid UserId,
    Guid VendorId,
    string Email,
    DateTimeOffset AccessTokenExpiresAt,
    string AccessToken,
    IReadOnlyList<string> Roles,
    IReadOnlyList<string> Permissions,
    Guid? CustomerId);

{
    "vendorName": "Nordic Software",
    "email": "admin@example.com",
    "password": "..."
}

```

Vendor login resolution:

```

VendorName
-> Platform DB
-> SOFTWARE_VENDOR
-> SoftwareVendor.Id
-> Resolve Vendor DB
-> Find USER_ACCOUNT by Email
-> Verify Password
-> Create authenticated session

```

Vendor Name constraint: SOFTWARE_VENDOR.Name must be unique in the Platform DB, trimmed before storage/comparison, and resolved case-insensitively. Renaming a Vendor changes the name users enter at login. SOFTWARE_VENDOR.Id remains the authoritative internal tenant identifier.

Refresh credentials: The exact transport and storage mechanism for refresh credentials is part of the Authentication/Authorization Flow design. The API contract reserves refresh endpoints without requiring the UI to know database implementation details.

5. Platform Vendor Administration

These endpoints operate at Platform scope and are available only to BridgeControl Platform Administrators.

Method	Route	Purpose	Authorization
GET	/api/v1/platform/vendors	List Vendors	PlatformAdmin
POST	/api/v1/platform/vendors	Create Vendor	PlatformAdmin
GET	/api/v1/platform/vendors/{vendorId}	Get Vendor	PlatformAdmin
PATCH	/api/v1/platform/vendors/{vendorId}	Edit Vendor	PlatformAdmin
POST	/api/v1/platform/vendors/{vendorId}/deactivate	Deactivate Vendor	PlatformAdmin
POST	/api/v1/platform/vendors/{vendorId}/reactivate	Reactivate Vendor	PlatformAdmin
POST	/api/v1/platform/vendors/{vendorId}/administrators	Create/invite initial Vendor administrator	PlatformAdmin

```
public sealed record CreateVendorRequest(  
    string Name);
```

```
public sealed record VendorResponse(  
    Guid Id,  
    string Name,  
    string Status,  
    DateTimeOffset CreatedAt,  
    DateTimeOffset? DeactivatedAt);
```

```
public sealed record CreateVendorAdministratorRequest(  
    string Email);
```

- Creating a Vendor provisions its isolated Vendor DB and Vendor File Archive.
- ConnectionSecretRef, AccessSecretRef, database names, and storage identifiers are infrastructure data and are never returned to ordinary UI clients.
- Creating an initial Vendor administrator does not allow the Platform Administrator to set or know that user's permanent password; activation is completed by the invited user.

6. User, Role and Access Administration

Method	Route	Purpose	Typical permission
GET	/api/v1/users	List accessible users	users.read
POST	/api/v1/users	Create/invite user	users.manage
GET	/api/v1/users/{userId}	Get user	users.read
PATCH	/api/v1/users/{userId}	Edit user	users.manage
POST	/api/v1/users/{userId}/deactivate	Deactivate user	users.manage
POST	/api/v1/users/{userId}/reactivate	Reactivate user	users.manage
PUT	/api/v1/users/{userId}/roles	Replace assigned roles	users.manage
GET	/api/v1/users/{userId}/customer-access	Get Technician Customer access	users.read
PUT	/api/v1/users/{userId}/customer-access	Set Technician Customer access	users.manage
GET	/api/v1/users/{userId}/installation-access	Get Customer User Installation access	users.read
PUT	/api/v1/users/{userId}/installation-access	Set Customer User Installation access	users.manage
GET	/api/v1/roles	List assignable roles	users.read
GET	/api/v1/permissions	List known permissions	users.read

```
public sealed record CreateUserRequest(  
    string Email,  
    Guid? CustomerId,  
    IReadOnlyList<Guid> RoleIds);
```

```
public sealed record UserResponse(  
    Guid Id,  
    Guid? CustomerId,  
    string Email,  
    string Status,  
    IReadOnlyList<string> Roles,  
    DateTimeOffset CreatedAt);
```

```
public sealed record SetUserRolesRequest(  
    IReadOnlyList<Guid> RoleIds);
```

```
public sealed record SetCustomerAccessRequest(  
    IReadOnlyList<Guid> CustomerIds);
```

```
public sealed record SetInstallationAccessRequest(  
    IReadOnlyList<Guid> InstallationIds);
```

- CustomerId = null represents a Vendor user; a populated CustomerId represents a Customer user.
- Vendor Administrators may administer users inside their own Vendor tenant.
- Customer Administrators may administer users only inside their own Customer.
- Technician Customer access cannot reference another Vendor.
- Customer User Installation access cannot reference an Installation owned by another Customer.
- Administrators invite/activate accounts; permanent passwords are set by the users themselves.

7. Customer Administration

Method	Route	Purpose	Typical permission
GET	/api/v1/customers	List accessible Customers	customers.read
POST	/api/v1/customers	Create Customer	customers.manage
GET	/api/v1/customers/{customerId}	Get Customer	customers.read
PATCH	/api/v1/customers/{customerId}	Edit Customer	customers.manage
POST	/api/v1/customers/{customerId}/deactivate	Deactivate Customer	customers.manage
POST	/api/v1/customers/{customerId}/reactivate	Reactivate Customer	customers.manage

```
public sealed record CreateCustomerRequest(  
    string Name);
```

```
public sealed record UpdateCustomerRequest(  
    string Name);
```

```
public sealed record CustomerResponse(  
    Guid Id,  
    string Name,  
    string Status,  
    DateTimeOffset CreatedAt);
```

Technician visibility: A Vendor Technician may only list or access Customers granted through `TECHNICIAN_CUSTOMER_ACCESS`.

8. Product Management

Method	Route	Purpose	Typical permission
GET	/api/v1/products	List products	products.read
POST	/api/v1/products	Create product	products.manage
GET	/api/v1/products/{productId}	Get product	products.read
PATCH	/api/v1/products/{productId}	Edit product	products.manage
POST	/api/v1/products/{productId}/deactivate	Deactivate product	products.manage
POST	/api/v1/products/{productId}/reactivate	Reactivate product	products.manage
GET	/api/v1/products/{productId}/deployment-settings	Read application deployment defaults	products.read
PUT	/api/v1/products/{productId}/deployment-settings	Set application deployment defaults	products.manage

```
public sealed record CreateProductRequest(  
    string Name,  
    string? Description);
```

```
public sealed record ProductResponse(  
    Guid Id,  
    string Name,  
    string? Description,  
    string Status,  
    DateTimeOffset CreatedAt);
```

- Application deployment defaults: product deployment settings contain Kind (Desktop or Service), ConfigurationPath, and either ExecutablePath (Desktop) or ServiceName (Service). Paths are absolute local Windows paths; no environment-variable expansion or caller-supplied archive paths are accepted. Desktop configuration targets the shared settings file; user settings may override it. Products.manage is required for writes; products.read for reads.
- public sealed record DeploymentSettings(string Kind, string ConfigurationPath, string? ExecutablePath, string? ServiceName);
 public sealed record DeploymentSettingsResponse(DeploymentSettings? Settings, InstallerIdentity? Installer, bool Frozen);
- GET and PUT return DeploymentSettingsResponse with 200. Settings may be null before configuration. Product settings apply to future releases and do not change published release metadata.

9. Product Version Management

Method	Route	Purpose	Typical permission
GET	/api/v1/products/{productId}/versions	List product versions	products.read
POST	/api/v1/products/{productId}/versions	Create draft version	products.manage
GET	/api/v1/products/{productId}/versions/{versionId}	Get version	products.read
PATCH	/api/v1/products/{productId}/versions/{versionId}	Edit draft version	products.manage
POST	/api/v1/products/{productId}/versions/{versionId}/publish	Publish version	products.manage
GET	/api/v1/products/{productId}/versions/{versionId}/deployment-settings	Read version deployment settings	products.read
PUT	/api/v1/products/{productId}/versions/{versionId}/deployment-settings	Set draft version deployment settings	products.manage
POST	/api/v1/products/{productId}/versions/{versionId}/deployment-settings/initialize	Initialize missing published deployment settings once	products.manage

```
public sealed record CreateProductVersionRequest(
    string Version);
```

```
public sealed record ProductVersionResponse(
    Guid Id,
    Guid ProductId,
    string Version,
    string Status,
    DateTimeOffset CreatedAt,
    DateTimeOffset? PublishedAt);
```

- Version values must be unique within a Product.
- A published Product Version is immutable for installer/schema-defining content.
- Publishing fails unless required MSI and JSON Schema artifacts are present and valid.
- Version deployment settings: draft settings may override product defaults; publication freezes an explicit snapshot, copying product defaults when no override exists. Published settings cannot be edited. Releases without settings remain available to legacy clients but require initialization before API-driven deployment.
- Existing-release initialization: POST .../deployment-settings/initialize accepts DeploymentSettings only for a published version. It may fill missing deployment settings and extract missing MSI metadata from the original verified published installer. It does not replace installer/schema bytes. An identical retry returns 200; attempts to change initialized settings return 409. Initialization is audited. The installer must have an UpgradeCode.
- Invalid settings return 400; missing or concealed resources return 404; publication or initialization conflicts return 409. Published application metadata is independent of user-editable JSON configuration values.

10. MSI and JSON Schema Management

The API exposes installer and schema resources explicitly even though persistence uses `FILE_ARTIFACT` and `PRODUCT_VERSION_ARTIFACT`. This keeps storage implementation details out of the client contract.

10.1 MSI installer

Method	Route	Purpose	Typical permission
PUT	/api/v1/products/{productId}/versions/{versionId}/installer	Upload/replace draft MSI	products.manage
GET	/api/v1/products/{productId}/versions/{versionId}/installer	Get installer metadata	products.read
GET	/api/v1/products/{productId}/versions/{versionId}/installer/content	Download installer	products.read
DELETE	/api/v1/products/{productId}/versions/{versionId}/installer	Remove draft installer	products.manage

10.2 JSON Schema

Method	Route	Purpose	Typical permission
PUT	/api/v1/products/{productId}/versions/{versionId}/schema	Upload/replace draft schema	products.manage
GET	/api/v1/products/{productId}/versions/{versionId}/schema	Get schema metadata	products.read
GET	/api/v1/products/{productId}/versions/{versionId}/schema/content	Download schema	products.read
DELETE	/api/v1/products/{productId}/versions/{versionId}/schema	Remove draft schema	products.manage

```
public sealed record FileArtifactResponse(  
    Guid Id,  
    string ArtifactType,  
    string FileName,  
    string ContentType,  
    string Sha256,  
    long SizeBytes,  
    DateTimeOffset CreatedAt);
```

- Uploads use multipart/form-data.
- The Backend computes the authoritative SHA-256 value.
- StorageIdentifier is internal and is not returned through the public DTO.
- Published version artifacts cannot be silently replaced.
- JSON Schema uploads must contain valid JSON and a supported valid JSON Schema.
- Installer identity: upload inspection extracts ProductCode, UpgradeCode (nullable for legacy packages) and MSI ProductVersion, storing them with the immutable artifact. These values are returned through the version deployment-settings response; clients never supply them. Managed deployments require an UpgradeCode. A release label is distinct from the MSI version string.
- public sealed record InstallerIdentity(Guid ProductCode, Guid? UpgradeCode, string Version);

11. Configuration Management

Configuration is exposed at the three levels defined by BridgeControl: Product Default, Customer Profile, and Installation Profile. Each accepted change creates a CONFIGURATION_REVISION; identical repeated PUT requests must resolve to the same current state without creating duplicate revisions.

11.1 Product Default configuration

Method	Route	Purpose
GET	/api/v1/products/{productId}/configuration	Get current Product Default configuration
PUT	/api/v1/products/{productId}/configuration	Set Product Default configuration
GET	/api/v1/products/{productId}/configuration/revisions	List Product Default revision history

11.2 Customer configuration

Method	Route	Purpose
GET	/api/v1/customers/{customerId}/products/{productId}/configuration	Get current Customer profile
PUT	/api/v1/customers/{customerId}/products/{productId}/configuration	Set Customer profile
GET	/api/v1/customers/{customerId}/products/{productId}/configuration/revisions	List Customer profile revisions

11.3 Installation configuration

Method	Route	Purpose
GET	/api/v1/installations/{installationId}/configuration	Get current Installation profile
PUT	/api/v1/installations/{installationId}/configuration	Set Installation profile
GET	/api/v1/installations/{installationId}/configuration/revisions	List Installation profile revisions

```
public sealed record UpdateConfigurationRequest(
    Guid ProductVersionId,
    JsonElement Values);

public sealed record ConfigurationRevisionResponse(
    Guid Id,
    Guid ProductVersionId,
    int RevisionNumber,
    JsonElement Values,
    Guid CreatedByUserId,
    DateTimeOffset CreatedAt);
```

- Values must validate against the JSON Schema belonging to ProductVersionId.
- Configuration references cannot cross Product, Customer, or Installation boundaries.
- Revision history is append-only; previous revisions are not overwritten.
- The Backend recalculates affected effective configurations when a source profile changes.

12. Effective Configuration

Effective configuration is the generated, merged configuration for an Installation. It is composed from the applicable Product Default, Customer, and Installation configuration revisions and is read-only to clients.

Method	Route	Purpose
GET	/api/v1/installations/{installationId}/effective-configuration	Get current effective revision
GET	/api/v1/installations/{installationId}/effective-configuration/revisions	List effective revision history
GET	/api/v1/installations/{installationId}/effective-configuration/revisions/{revisionId}	Get a specific effective revision

```
public sealed record EffectiveConfigurationResponse(  
    Guid RevisionId,  
    Guid InstallationId,  
    Guid ProductVersionId,  
    int RevisionNumber,  
    JsonElement Values,  
    IReadOnlyList<Guid> SourceRevisionIds,  
    DateTimeOffset CreatedAt);
```

Effective configuration lineage is represented by
EFFECTIVE_CONFIGURATION_SOURCE_REVISION.

13. Configuration Migration

Method	Route	Purpose	Typical permission
GET	/api/v1/products/{productId}/configuration-migrations	List migration definitions	configuration.read
POST	/api/v1/products/{productId}/configuration-migrations	Create migration definition	configuration.manage
GET	/api/v1/products/{productId}/configuration-migrations/{migrationId}	Get migration definition	configuration.read
PUT	/api/v1/products/{productId}/configuration-migrations/{migrationId}	Replace migration definition	configuration.manage
POST	/api/v1/installations/{installationId}/configuration/migration-preview	Preview migration	configuration.manage
POST	/api/v1/installations/{installationId}/configuration/migrate	Apply approved migration	configuration.manage

```
public sealed record MigrationRuleRequest(
```

```
    int Sequence,  
    string RuleType,  
    string? SourcePath,  
    string? TargetPath,  
    string? Transform,  
    bool IsDestructive);
```

```
public sealed record CreateConfigurationMigrationRequest(
```

```
    Guid FromProductVersionId,  
    Guid ToProductVersionId,  
    IReadOnlyList<MigrationRuleRequest> Rules);
```

```
public sealed record ConfigurationMigrationPreviewRequest(
```

```
    Guid TargetProductVersionId);
```

```
public sealed record ApplyConfigurationMigrationRequest(
```

```
    Guid TargetProductVersionId,  
    IReadOnlyList<Guid> ApprovedDestructiveRuleIds);
```

- Source and target Product Versions must belong to the same Product.
- Rule Sequence values must be unique within the migration.
- The source configuration is validated against its existing schema before migration.
- The migrated configuration is validated against the target schema before it can become effective.
- Destructive rules require explicit approval.
- Unresolved values cause migration to fail rather than producing an incomplete configuration.

14. Installation Management

Method	Route	Purpose	Typical permission
GET	/api/v1/installations	List accessible Installations	installations.read
GET	/api/v1/installations/{installationId}	Get Installation	installations.read
PATCH	/api/v1/installations/{installationId}	Edit manageable properties	installations.manage
POST	/api/v1/installations/{installationId}/deactivate	Deactivate Installation	installations.manage
POST	/api/v1/installations/{installationId}/reactivate	Reactivate Installation	installations.manage
POST	/api/v1/installations	Assign product to an enrolled Agent	installations.manage

Collection filters may be combined with pagination, for example:

```
GET /api/v1/installations?customerId={id}&productId={id}&status=online&page=1&pageSize=25
```

```
public sealed record InstallationResponse(  
    Guid Id,  
    Guid CustomerId,  
    Guid ProductId,  
    Guid? AgentId,  
    string Name,  
    string Status,  
    DateTimeOffset RegisteredAt,  
    DateTimeOffset? LastSeenAt);
```

Creation: An authorized UI user can assign a Product to an existing active Agent through POST /api/v1/installations. The Backend derives CustomerId from that Agent. Agent registration through POST /api/v1/agent/installations remains supported for older agents; it is not required for centrally assigned installations.

- public sealed record AssignProductRequest(Guid AgentId, Guid ProductId, string Name);
- Assignment returns 201 and Location /api/v1/installations/{id}, including on an identical retry. The same Agent/Product returns its existing active installation; disabled or ambiguous existing assignments return 409. The Agent, Customer and Product must be active, and a published release must have deployment settings. Missing or out-of-scope resources return 404.
- Assignment requires installations.manage and a Vendor Administrator, scoped Vendor Technician or Customer Administrator role. Customer Users with individual installation grants cannot create assignments. Ownership is resolved from the Agent and cannot be supplied or changed by the request. Registration names remain stable independently of display-name changes.
- Assignment does not install software immediately. The existing license assignment, effective-configuration generation and desired-state operations authorize deployment. The first Agent observation may report NotInstalled; no installed version or heartbeat is invented when assigning.

15. Licensing

Method	Route	Purpose	Typical permission
GET	/api/v1/licenses	List licenses	licenses.read

POST	/api/v1/licenses	Create license	licenses.manage
GET	/api/v1/licenses/{licenseId}	Get license	licenses.read
PATCH	/api/v1/licenses/{licenseId}	Edit license	licenses.manage
POST	/api/v1/licenses/{licenseId}/deactivate	Deactivate license	licenses.manage
POST	/api/v1/licenses/{licenseId}/reactivate	Reactivate license	licenses.manage
PUT	/api/v1/installations/{installationId}/license	Assign license	licenses.manage
DELETE	/api/v1/installations/{installationId}/license	Remove license assignment	licenses.manage

```
public sealed record CreateLicenseRequest(
```

```
    Guid CustomerId,  
    Guid ProductId,  
    string LicenseReference,  
    DateTimeOffset ValidFrom,  
    DateTimeOffset? ValidUntil,  
    int? MaxInstallations);
```

```
public sealed record AssignLicenseRequest(
```

```
    Guid LicenseId);
```

```
public sealed record LicenseResponse(
```

```
    Guid Id,  
    Guid CustomerId,  
    Guid ProductId,  
    string LicenseReference,  
    string Status,  
    DateTimeOffset ValidFrom,  
    DateTimeOffset? ValidUntil,  
    int? MaxInstallations);
```

- A License assigned to an Installation must belong to the same Customer and Product.
- A license with MaxInstallations cannot be assigned beyond its allowed active Installation count.
- Expired or deactivated licenses cannot authorize new desired-state deployment operations.

16. Desired and Reported Installation State

16.1 Desired state

Method	Route	Purpose	Typical permission
GET	/api/v1/installations/{installationId}/desired-state	Read desired state	installations.read
PUT	/api/v1/installations/{installationId}/desired-state	Set desired state	installations.manage

```
public sealed record SetDesiredStateRequest(  
    Guid ProductVersionId,  
    Guid EffectiveConfigurationRevisionId);
```

```
public sealed record DesiredStateResponse(  
    Guid InstallationId,  
    Guid ProductVersionId,  
    Guid EffectiveConfigurationRevisionId,  
    DateTimeOffset UpdatedAt);
```

- The Product Version must belong to Installation.Product.
- The Effective Configuration Revision must belong to the Installation and match the selected Product Version.
- The Installation must have a valid applicable License.
- Changing desired state may create a Deployment.

16.2 Reported state

Method	Route	Purpose	Authorization
GET	/api/v1/installations/{installationId}/reported-state	Read latest Agent-reported state	Authorized UI user
PUT	/api/v1/agent/installations/{installationId}/reported-state	Update Agent-reported state	Authenticated Agent

```
public sealed record ReportedStateResponse(  
    Guid InstallationId,  
    Guid? InstalledProductVersionId,  
    Guid? EffectiveConfigurationRevisionId,  
    string? ConfigurationHash,  
    string Status,  
    DateTimeOffset ReportedAt);
```

```
public sealed record ReportInstallationStateRequest(  
    Guid? InstalledProductVersionId,  
    Guid? EffectiveConfigurationRevisionId,  
    string? ConfigurationHash,  
    string Status,  
    DateTimeOffset ReportedAt);
```

Agent PUT semantics: Reported-state updates are idempotent. Retransmitting the same state must not create duplicate logical state.

- Desktop state: Ready means the MSI version and executable exist and the stored configuration matches the applied receipt. It requires InstalledProductVersionId, EffectiveConfigurationRevisionId and ConfigurationHash. It does not mean a user interface is running. Matching Ready evidence reconciles a desktop deployment; service applications still report Running. Stopped and RestartRequired do not count as matching.

17. Deployment Status and History

Method	Route	Purpose	Typical permission
GET	/api/v1/deployments	Search deployments	deployments.read
GET	/api/v1/deployments/{deploymentId}	Get deployment details	deployments.read
GET	/api/v1/installations/{installationId}/deployments	Get Installation deployment history	deployments.read
POST	/api/v1/deployments/{deploymentId}/cancel	Cancel pending deployment	deployments.manage

```
public sealed record DeploymentResponse(  
    Guid Id,  
    Guid InstallationId,  
    Guid TargetProductVersionId,  
    Guid? TargetConfigurationRevisionId,  
    string OperationType,  
    string Status,  
    DateTimeOffset RequestedAt,  
    DateTimeOffset? CompletedAt);  
  
public sealed record DeploymentAttemptResponse(  
    Guid Id,  
    int AttemptNumber,  
    string Status,  
    string? ErrorCode,  
    string? ErrorMessage,  
    DateTimeOffset StartedAt,  
    DateTimeOffset? CompletedAt);
```

Detailed deployment responses include their DEPLOYMENT_ATTEMPT history.

18. Agent Enrollment and Authentication

Agent bootstrap is separated from ordinary user authentication. An authorized user creates a short-lived enrollment, which a new Agent exchanges for its registered identity and long-term credential material.

Method	Route	Purpose	Authorization
POST	/api/v1/agent-enrollments	Create short-lived Agent enrollment	agents.manage
POST	/api/v1/agent/register	Register Agent using enrollment	Enrollment credential
POST	/api/v1/agent/auth/token	Obtain Agent access token	Agent credential

```
public sealed record CreateAgentEnrollmentRequest(  
    Guid CustomerId);
```

```
public sealed record AgentEnrollmentResponse(  
    string EnrollmentCode,  
    DateTimeOffset ExpiresAt);
```

```
public sealed record RegisterAgentRequest(  
    string EnrollmentCode,  
    string MachineName,  
    string AgentVersion);
```

Persistence impact: Secure Agent bootstrap requires an AGENT_ENROLLMENT persistence concept in addition to AGENT and AGENT_CREDENTIAL. The final credential format and rotation rules are defined in the Agent/Auth flow design.

19. Agent Communication API

19.1 Heartbeat

Method	Route	Purpose
POST	/api/v1/agent/heartbeat	Report Agent liveness, version, and status

```
public sealed record AgentHeartbeatRequest(  
    string AgentVersion,  
    string Status,  
    DateTimeOffset ReportedAt);
```

19.2 Managed Installations

Method	Route	Purpose
GET	/api/v1/agent/installations	List Installations managed by the authenticated Agent
POST	/api/v1/agent/installations	Register a managed software Installation
GET	/api/v1/agent/installations/{installationId}/desired-state	Retrieve desired state
PUT	/api/v1/agent/installations/{installationId}/reported-state	Report actual state

```
public sealed record RegisterInstallationRequest(  
    Guid ProductId,  
    string Name,  
    string? InstalledVersion);
```

- The Backend automatically associates the Installation with the authenticated Agent and its Customer.
- An Agent cannot read or update Installations assigned to another Agent unless an explicit reassignment workflow permits it.
- All Agent requests remain constrained to the resolved Vendor tenant.
- API-driven discovery: GET /api/v1/agent/installations retains the paged envelope and existing installation fields, adding Application. Each Agent sees only its active owned installations for active Products. Application is null when compatible published deployment metadata is missing; the Agent must wait for central setup. No separate assignment-discovery route is introduced.
- ```
public sealed record AgentVersionBinding(Guid ProductVersionId, Guid ProductCode, string Version);
public sealed record AgentApplication(Guid ProductId, Guid SettingsVersionId, string Name, string Kind, Guid UpgradeCode, string ConfigurationPath, string ExecutablePath, string ServiceName, IReadOnlyList<AgentVersionBinding> Versions);
public sealed record AgentInstallationAssignment(Guid Id, Guid CustomerId, Guid ProductId, Guid? AgentId, string Name, string Status, DateTimeOffset RegisteredAt, DateTimeOffset? LastSeenAt, AgentApplication? Application);
```
- Application.Name is the stable registration identity. Application settings come from the desired published version, or the latest configured published version before desired state is selected. Versions contains published MSI identities in the selected UpgradeCode family. Download access still requires the existing authorized desired-state and artifact checks.

- Agents discover all assignment pages before processing work, validate ownership and local paths, and persist installation IDs and a settings snapshot for recovery. Explicit legacy local mappings remain supported during rollout. A centrally configured assignment replaces a local mapping for the same Product. New setups require only the HTTPS server address and enrollment identity.

## 20. Agent Artifact Downloads

| Method | Route                                        | Purpose                                           | Authorization       |
|--------|----------------------------------------------|---------------------------------------------------|---------------------|
| GET    | /api/v1/agent/artifacts/{artifactId}/content | Download an authorized MSI/configuration artifact | Authenticated Agent |

The Backend verifies that the requested artifact is required by an Installation managed by the authenticated Agent. Artifacts unrelated to the Agent's authorized Installations are not exposed.

### Authenticated Agent

- > Installation
- > Desired Product Version / Configuration
- > Authorized artifact
- > Vendor File Archive

## 21. Agent Deployment Execution and Reporting

| Method | Route                                                              | Purpose                                           |
|--------|--------------------------------------------------------------------|---------------------------------------------------|
| GET    | /api/v1/agent/deployments?status=pending                           | Retrieve pending work for the authenticated Agent |
| POST   | /api/v1/agent/deployments/{deploymentId}/attempts                  | Start a deployment attempt                        |
| PATCH  | /api/v1/agent/deployments/{deploymentId}/attempts/{attemptId}<br>} | Report attempt progress/result                    |

```
public sealed record StartDeploymentAttemptResponse(
 Guid AttemptId,
 int AttemptNumber);
```

```
public sealed record UpdateDeploymentAttemptRequest(
 string Status,
 string? ErrorCode,
 string? ErrorMessage,
 DateTimeOffset? CompletedAt);
```

- Attempt numbers are generated by the Backend.
- The Agent cannot report against a Deployment belonging to another Agent/Installation.
- Completed attempts are immutable except for explicitly supported correction/audit workflows.
- Failed attempts preserve error code/message and timestamps.
- Desktop execution policy: user acceptance is temporarily assumed. The Agent closes the exact configured executable before install/update/configuration, may terminate it if graceful closure fails, and never launches the desktop in the LocalSystem service session. Startup remains controlled by the application. User prompts and postponement through a session companion are future work.

## 22. Logs and Monitoring

### 22.1 Agent log submission

| Method | Route              | Purpose                                          |
|--------|--------------------|--------------------------------------------------|
| POST   | /api/v1/agent/logs | Submit Agent/Installation log entries in a batch |

```
public sealed record AgentLogEntryRequest(
 Guid? InstallationId,
 string Severity,
 string Source,
 string Message,
 DateTimeOffset CreatedAt);

public sealed record SubmitAgentLogsRequest(
 IReadOnlyList<AgentLogEntryRequest> Entries);
```

### 22.2 Administrative monitoring

| Method | Route                                       | Purpose                   | Typical permission |
|--------|---------------------------------------------|---------------------------|--------------------|
| GET    | /api/v1/logs                                | Search accessible logs    | logs.read          |
| GET    | /api/v1/installations/{installationId}/logs | Installation logs         | logs.read          |
| GET    | /api/v1/agents                              | List Agents               | agents.read        |
| GET    | /api/v1/agents/{agentId}                    | Get Agent status          | agents.read        |
| GET    | /api/v1/agents/{agentId}/logs               | Agent logs                | logs.read          |
| GET    | /api/v1/agents/{agentId}/heartbeats         | Heartbeat history         | agents.read        |
| GET    | /api/v1/monitoring/overview                 | Aggregate status overview | monitoring.read    |

**Overview endpoint:** The monitoring overview is a read-model convenience endpoint. It does not replace the underlying Installation, Agent, Deployment, License, and Log resources.

## 23. Authorization Model

BridgeControl uses roles to group permissions, while endpoint authorization is evaluated using permissions plus tenant and resource boundaries.

| Permission family | Examples                                       |
|-------------------|------------------------------------------------|
| Platform          | platform.vendors.read, platform.vendors.manage |
| Users             | users.read, users.manage                       |
| Customers         | customers.read, customers.manage               |
| Products          | products.read, products.manage                 |
| Installations     | installations.read, installations.manage       |
| Configuration     | configuration.read, configuration.manage       |
| Licensing         | licenses.read, licenses.manage                 |
| Deployments       | deployments.read, deployments.manage           |
| Agents            | agents.read, agents.manage                     |
| Logs/Monitoring   | logs.read, monitoring.read                     |

Every protected request is evaluated using the following layers:

### Authentication

+ Permission  
+ Vendor tenant boundary  
+ Customer boundary  
+ Technician / Installation access assignment  
= Authorized request

- A permission never grants cross-Vendor access.
- A Customer User cannot access resources belonging to another Customer.
- A Vendor Technician is constrained by `TECHNICIAN_CUSTOMER_ACCESS`.
- A Customer User is constrained by `CUSTOMER_USER_INSTALLATION_ACCESS` where applicable.
- Platform Administrator endpoints are separated under `/api/v1/platform`.

## 24. Error and ProblemDetails Model

Errors are returned as application/problem+json using RFC Problem Details.

### 24.1 Validation example

```
{
 "type": "https://bridgecontrol/errors/validation",
 "title": "One or more validation errors occurred.",
 "status": 400,
 "traceId": "00-abcd...",
 "errors": {
 "email": [
 "A valid email address is required."
]
 }
}
```

### 24.2 Business conflict example

```
{
 "type": "https://bridgecontrol/errors/configuration-migration-required",
 "title": "Configuration migration is required.",
 "status": 409,
 "detail": "The selected product version uses a different configuration schema."
}
```

**Cross-tenant lookups:** When revealing that a resource exists would disclose another tenant's data, the API returns 404 rather than confirming existence with 403.

## 25. Standard HTTP Status Codes

| Status                     | Meaning in BridgeControl                            |
|----------------------------|-----------------------------------------------------|
| 200 OK                     | Successful read/update with response body           |
| 201 Created                | Resource successfully created                       |
| 204 No Content             | Successful operation with no response body          |
| 400 Bad Request            | Invalid input or validation failure                 |
| 401 Unauthorized           | Missing, invalid, or expired authentication         |
| 403 Forbidden              | Authenticated but not permitted                     |
| 404 Not Found              | Resource absent or intentionally hidden from caller |
| 409 Conflict               | Business/current-state conflict                     |
| 413 Payload Too Large      | Uploaded payload exceeds configured limit           |
| 415 Unsupported Media Type | Unsupported upload/body content type                |
| 429 Too Many Requests      | Rate limit reached                                  |
| 500 Internal Server Error  | Unexpected server failure                           |

## 26. Core Validation Rules

- Guid.Empty is never a valid resource identifier.
- Required strings are trimmed and cannot be empty.
- Email addresses are normalized before uniqueness checks.
- User email uniqueness is enforced within the appropriate isolated Vendor DB.
- SOFTWARE\_VENDOR.Name is unique in the Platform DB and resolved case-insensitively for login.
- Product Version values are unique within a Product.
- Published Product Versions cannot have their MSI or JSON Schema silently replaced.
- Installer uploads must satisfy the configured MSI file validation rules.
- JSON Schema uploads must contain valid supported JSON Schema.
- Configuration values must validate against the selected Product Version schema.
- Configuration references cannot cross Product, Customer, or Installation ownership boundaries.
- Migration source and target versions must belong to the same Product.
- Migration Sequence values are unique within a migration.
- Destructive migration rules require explicit approval.
- Licenses cannot be assigned across Customers or Products.
- Agent operations are restricted to the authenticated Agent's resolved Vendor, Customer, and managed Installations.
- Customer users cannot be granted Installation access outside their own Customer.
- Server-generated timestamps and hashes are authoritative where applicable.

## 27. OpenAPI and Swagger

The ASP.NET Core Backend API generates an OpenAPI document from the implemented contract. The OpenAPI definition is the authoritative technical API description and should remain synchronized with the code.

- Every endpoint has a stable operation identifier and a resource-based OpenAPI tag.
- Request and response DTO schemas are included.
- Bearer authentication requirements are declared.
- Path, query, and body parameters are documented with required/optional status.
- Success and expected error status codes are documented.
- ProblemDetails response schemas are referenced consistently.
- File upload endpoints declare multipart/form-data.
- File download endpoints declare appropriate binary content types.
- Endpoint summaries and descriptions explain authorization-relevant behavior.

| Resource     | Path             |
|--------------|------------------|
| OpenAPI JSON | /openapi/v1.json |
| Swagger UI   | /swagger         |

## 28. Persistence Considerations Identified by the API

The API design uses the existing ER model directly wherever possible. Two authentication/bootstrap concerns require persistence that is not represented completely by the current domain diagrams.

1. Agent enrollment: secure bootstrap requires an AGENT\_ENROLLMENT concept with a short lifetime, single-use state, Customer/Vendor association, and audit data.
2. Session and recovery state: refresh sessions, activation tokens, and password-reset tokens require persistence or another secure server-side mechanism. Their exact schema is defined during Authentication/Authorization Flow design.

**No TenantKey:** No TenantKey is required. SOFTWARE\_VENDOR.Id is the internal tenant identifier and SOFTWARE\_VENDOR.Name is used to identify the Vendor during login.

- API-managed applications upgrade: the AddApiManagedApplications vendor migration adds nullable Product and ProductVersion deployment-settings JSON plus installer-identity JSON on FileArtifact. Existing records are preserved. Apply the migration through the offline vendor upgrade workflow before starting the new API. Existing published releases use the explicit audited initialization endpoint; never silently rewrite published installer/schema content. Upgrade the API before agents (0.3.0); preserve enrollment identity and deployment history.

## 29. Completion Criteria for the Initial API Design

The initial API design is considered ready for implementation when:

- The endpoint routes and resource boundaries are accepted.
- The authentication and Agent bootstrap flows are finalized.
- Permissions are mapped to the agreed BridgeControl roles.
- DTO validation constraints are implemented consistently.
- OpenAPI generation exposes the documented contract.
- Tenant, Customer, and Installation boundaries are enforced server-side.
- The API can be exercised through Swagger without direct database or file-archive access by clients.