BioAPI™ Scope

Background

The biometrics marketplace is fragmented with many competing “standard” biometrics APIs. Each API targets an application space (simple high level programming interface, virtual device support, layered API). However, many of the existing APIs are fingerprint-centric and/or lack sufficient detail to drive systems required for speaker and face recognition. Additionally, the multitude of standards causes confusion for application writers and adds a tremendous work load for biometrics vendors who must provide toolkits to aid application development under multiple APIs.

The founding companies of the BioAPI Consortium recognized the need for a single comprehensive biometrics API to ensure widespread rollout of biometrics technologies. Hence, Compaq, IBM, Identicator, Microsoft, Miros, and Novell founded the BioAPI™ consortium in April 1998. This document addresses the scope of biometrics standardization and briefly discusses deliverables of the BioAPI Consortium. For the most updated information on consortium activities and status, please visit the BioAPI web site at http://www.bioapi.org.

BioAPI Industry Value

Need for Another Biometrics API

Given the market confusion caused by multiple competing biometrics APIs, does the industry need yet another biometrics API?

In answering this question, the Consortium decided to first define the key design and development drivers essential for a comprehensive biometrics API. This analysis led to the following:

1. Provide a multi-level API, addressing a wide variety of implementation environments
2. Provide common framework supporting multiple biometric technologies
3. Provide a robust security architecture for biometrics
4. Provide a vendor independent process for development and ownership

Table 1 -- Drivers for a Comprehensive API

Support for multiple API levels, ensures that the BioAPI will meet the needs of industry security administrators, systems integrators, value add resellers, application developers and end users of biometrics technologies. The goal is for developers to use the highest level API that fulfills their software requirements. A common plug & play framework for multiple biometrics technologies ensures software investments are preserved across fingerprint, hand, voice, face, iris, and other biometrics. Security hooks built into the interface ensure data privacy throughout the stages of biometrics processing on the local machine and across distributed processing environments. The BioAPI development process must be open and not driven or owned by any single company.

The BioAPI Consortium then examined existing biometrics APIs as a starting point for the BioAPI specification. These APIs were measured with respect to their level of support for the development and design drivers of Table 1 above. While all available APIs addressed some of the requirements, none provided a consistent high level of support of all the design and development drivers. As a result, the
Consortium decided to proceed with a new design, which would leverage the positive aspects of each existing API to rapidly deliver a unified API to the biometrics industry.

### BioAPI Deliverables

A typical biometrics verification\(^1\) or identification\(^2\) system comprises multiple functional components:
- Capture device for raw biometric data
- APIs for biometrics aware applications and service providers
- Data processing engines/algorithms
- Matching/scoring model, an underlying security model
- Data storage services for reference data sets.

Each of these components can benefit from standardization. However key components like biometric processing engines and algorithms have serious intellectual property implications and are not appropriate for standardization. The remainder of this document will address the magnitude of the biometrics standardization effort from the BioAPI consortium.

The BioAPI consortium’s primary goal is standardization of a biometric independent multi-level API to foster widespread adoption of biometrics for verification and identification. API standardization allows developers to write software once that will work across biometrics technologies and across multi-vendor products within a biometrics technology with minimal, if any, changes. Multiple levels of API offer flexibility for application level programmers, whether they require rich low-level control of the verification/identification process or wish to program at a high level of abstraction. The BioAPI specification will define common biometrics terminology.

Past standardization efforts have bypassed security definitions. The BioAPI specification will ensure data movement among elements of the BioAPI architecture is secure and error free. However, BioAPI must remain independent of the underlying security framework. Finally, the BioAPI specification will define a run-time (i.e. framework) capable of supporting a comprehensive set of biometrics technologies: face, fingerprint, hand, iris, voice, and other emerging technologies.

The combination of a standard API and run-time facilitates development of interchangeable and interoperable products from multiple vendors. As a result, customers are not locked into a particular vendor or biometrics technology. Likewise biometrics vendors can develop modules that seamlessly integrate into the BioAPI plug-and-play framework and span multiple platforms, with little additional development. These deliverables are addressed below in more detail.

### BioAPI Goals and Requirements

Above we outlined the high level deliverables of the BioAPI in response to industry drivers for a comprehensive biometrics API. What does a "comprehensive" API and framework entail? Consider the following list of goals and requirements driving development of the BioAPI:

- Support multiple programming languages (i.e. C, C++, and Java)
- Provide an extensible framework for supporting most biometric devices and data types and allow deployment of cross-platform biometrics solutions.
  - BioAPI must not rely on any platform-specific service.
  - As much UI independence as is practical
- Support internal and external data storage models
- Provide support for multiple sources of raw biometric data

---

\(^1\) Biometrics verification is the process of 1-to-1 matching. Typically the reference biometric sample is indexed with some other data to reduce the sample size of the search.

\(^2\) Biometrics identification is the process of 1-to-many matching. Typically in this case the only input for a match is the biometric itself and it is compared across all reference biometrics data.
• Allow for multiple transformations on biometric input data, resulting in a final biometric template/model
• Support multi-process and multi-machine implementations.
• Provide tagging of data objects but not specify content (content remains vendor specific)
• Enable internationalization of BioAPI applications
• Provide a comprehensive scoring model (boolean/values, thresholds, etc.)
• Provide introspection mechanism for system meta-data (i.e. configuration) so that an application programmer can find out the structure of his/her environment programmatically.

**High Level Architecture and Terminology**

The BioAPI architecture is based on object abstractions of the key components within a biometric software system. These subsystem component abstractions can model simple to quite complex biometric systems. To aid in understanding the BioAPI architecture, consider the BioAPI object hierarchy and terminology depicted in Figure 1 below, followed by the object descriptions.

![Figure 1 -- BioAPI Object Level Components](image_url)

**BioAPI object level descriptions**

- **Biometric Space**: the scope/container of operation for a biometrics application.
- **Biometric System**: software and hardware (abstraction) system that implements part or all of the BioAPI. A biometric system is composed of one or more of the following objects:
  - **Source**: a logical input device (i.e. this can be a physical device or an input data stream from a file). A biometric system can contain multiple sources of various types.
  - **Transformer**: operates on a source to “transform” data into another kind of biometric object. Think of transformers as abstractions for typical biometrics functions such as feature extraction, compression, and encryption. These transformers can be linked together to handle virtually any real-world processing scenario.
  - **Registry**: data store for biometric objects. The application is isolated from the details of how or where data resides (local or remote).
• **Transporter**: allows secure biometrics data handling between biometrics systems. Transporters are tied into native OS or third party add-on security services, while providing a constant programming model for BioAPI applications.

• **Matcher**: standard call interface for verification and identification.

**Multi-level APIs**

BioAPI will provide multiple levels of API, from simple to complex, yet flexible. The reason for offering a multi-level API is to allow a standard interface for control of biometrics devices and the processing stages involved in performing a verification or verification function. As of the writing of this document the total number of levels is not finalised. However, for the present architecture discussion, assume four levels of API as shown in the BioAPI architectural diagram, Figure 2 below on page 4. Level 1 provides the greatest granularity of control to the application. Higher levels of API provide more abstraction of the underlying device or environment dependencies. Consequently, the programming impacts when changing biometric vendors or between biometric technologies is reduced accordingly at each higher API level. For this reason it is recommended that applications be developed to the highest level of API that provides the required functionality.

**Extensible Framework**

The multiple API levels are exposed within an extensible framework, supporting a comprehensive set of biometrics, including face, fingerprint, voice, hand recognition, iris and others. Figure 2 below details the multi-level API within a runtime environment that provides a plug-and-play framework for interchangeability of devices or algorithms.

![Figure 2 -- BioAPI Architecture](image)

The BioAPI specification will define a runtime consisting of the following components:

• **Framework Support Library (FSL)** -- internal library to support the runtime levels

• **Programming support Library (PSL)** -- utility library for vendors and application programmers
- **Hollow BioAPI interfaces** -- applications call are linked to and call into the runtime, which allows for multiple implementations of underlying interfaces at each API level.
- **Vendor Support Library (VSL)** -- support library for biometric vendors offering a standard vehicle for registering and removing biometrics and sources from the BioAPI configuration database.
- **Platform Abstraction Library (PAL)** -- public library to hide platform specific calls behind a standard set of interfaces. Examples of the abstractions are: file create/read/write, memory allocation/deallocation, mutex support for thread synchronization, UID operations, time query/format, and thread creation/management.

**Closing Remarks on BioAPI**

The multi-level API and runtime component allows for separation of biometric services from the implementation of the services. The multi-level API approach allows vendors to structure a specific implementation such that it can be broken down in the future. If API levels are thorough and well defined, then the industry benefits via standardization of functions at various levels of granularity.

Details of each component of the BioAPI framework will be forthcoming in the BioAPI specification and architecture white papers.

---

3 Hollow interfaces are part of the upper runtime component defined in the BioAPI specification. The hollow runtime interfaces route application requests to vendor supplied modules that implement the actual BioAPI functions.

4 A BioAPI interface is a set of related functions and is the basic level of modularity available to biometric vendors. Each API level contains a set of interfaces.

5 PAL provides platform independent OS services used primarily by vendor and framework components. Examples of these services are threads, mutexes, file handling, memory services, etc.