



# ActiveVOS Versioning

## Best Practices

Version: 1.0

**AN ACTIVE ENDPOINTS TECHNICAL NOTE**

© 2010 Active Endpoints Inc. ActiveVOS is a trademark of Active Endpoints, Inc. All other company and product names are the property of their respective owners.



2010

## Content

Introduction.....	3
Compatible Changes.....	3
Incompatible Changes.....	4
Versioning Strategies.....	5
Strict Versioning (New Change, New Contract).....	5
Flexible Versioning (Backwards Compatibility).....	6
Loose Versioning (Backwards and Forwards Compatibility).....	7
Versioning Contracts.....	7
How to Version a Schema document.....	8
How to Version a WSDL document.....	9
Artifact Handling Best Practices.....	11
Use of Relative Paths.....	11
Use of a Common Repository Project for Shared Artifacts.....	11
Understanding ActiveVOS Process Versioning and Revisions.....	12
ActiveVOS Deployment Considerations.....	13
Criteria for Deploying a Process Revision.....	13
Use of Common Business Process Archives.....	14
Avoiding the Deployment of WSDL Documents of the Same Namespace and WSDL Service QName to the ActiveVOS Resource Catalog.....	14
Relative Paths for Portability.....	15
Governing Contracts.....	16
Proactive Governance.....	16
Reactive Governance.....	16
Versioning ActiveVOS Projects.....	17
Refactoring Service Provider Projects.....	17
Refactoring Service Consumer Projects.....	17
Versioning POJO projects.....	18
About Active Endpoints.....	19

## Introduction

After a release is deployed, service consumers invoke provider services using the WSDL and schema contracts specified at design time and subsequently deployed to the ActiveVOS Resource Catalog. Months later when a new release is deployed there will likely be changes in the contracts. Some changes might not affect existing consumers which will continue using old contract versions, but other changes will break the communication between old consumers and the new service providers. The challenge: how to deploy new releases without breaking old consumers? More specifically, how do you avoid deploying new WSDL and schema definitions to the ActiveVOS Resource Catalog that break existing process definitions because a schema change has been introduced that breaks a prior (and running) version of a process?

This document describes versioning considerations and provides guidance. Readers are encouraged to read the “Versioning Schemas and WSDLs” Active Endpoints Best Practice document that you can find in the [“Developers Best Practices – Artifact Design and Versioning”](#) section of the Active Endpoints website. This document provides a more in-depth discussion of the issues. This companion to this document proposes specific best practices for versioning WSDL and XML Schema documents. It covers major and minor update concepts, recommendations for versioning these artifacts, and the impact of doing so.

## Compatible Changes

A change to a service contract that does not negatively affect its existing consumers is considered to a compatible change. This type of change does not impact existing consumers that consume old message formats, nor does it affect future consumers that can be designed to optionally send the new message format. Examples of compatible change include:

### Compatible Schema Changes

- Changing an element from required to optional
- adding a new optional XML Schema element or attribute declaration to a message definition
- adding a new XML Schema wildcard to a message definition type

### Compatible WSDL Changes

- adding a new WSDL operation definition and associated message definitions
- adding a new WSDL port type definition and associated operation definitions
- adding new WSDL binding and service definitions
- adding a new optional WS-Policy assertion
- adding a new WS-Policy alternative

### **Compatible BPEL Changes**

- Change in business logic
- Change in expressions

For a more in-depth discussion, please review the “Versioning Schemas and WSDLs” Active Endpoints Best Practice document.

## **Incompatible Changes**

If after a change a contract is no longer compatible with consumers it is considered to be an incompatible change. These are the types of changes that can break an existing contract and therefore impose the most challenges when it comes to versioning.

### **Incompatible Schema Changes**

- adding a new required XML Schema element or attribute declaration to a message definition
- increasing the constraint granularity (removing a wildcard) of an XML Schema element or attribute declaration of a message definition
- renaming an optional or required XML Schema element or attribute in a message definition
- removing an optional or required XML Schema element or attribute or wildcard from a message definition

### **Incompatible WSDL Changes**

- renaming an existing WSDL operation definition
- removing an existing WSDL operation definition
- changing the message exchange pattern (MEP) of an existing WSDL operation definition

- adding a fault message to an existing WSDL operation definition
- adding a new required WS-Policy assertion or expression
- adding a new ignorable WS-Policy expression (most of the time)

#### **Incompatible BPEL Changes**

- Changing a adding a new correlation set

For a more in-depth discussion, please review the “Versioning Schemas and WSDLs” Active Endpoints Best Practice document.

## **Versioning Strategies**

There is no de facto versioning technique and prescriptive guidance that can be offered for the WSDL, XML Schema, and WS-Policy definitions that comprises service contracts that will meet the needs of all. What we offer are three broad versioning strategies: strict, flexible, and loose. Each strategy offers its own benefits.

### **Strict Versioning (New Change, New Contract)**

The simplest approach to versioning a service contract is to require that a new version of a contract be defined whenever any kind of change is made to any part of the contract.

This is commonly implemented by changing the target namespace of a WSDL definition (and possibly the XML Schema definition) every time a (compatible or incompatible) change is made to the WSDL, XML Schema, or WS-Policy content related to the contract. The target namespace is used to identify a version instead of using a version attribute because changing the namespace value automatically mandates a change to all consumers. Each will now need to use the new version of the schema that defines new message types for example.

The strict approach may not always be practical. It is nonetheless explicit and sometimes warranted. Because both compatible and incompatible changes will engender a new contract version this approach is neither backwards nor forwards compatible.

A benefit of this strategy is its ability to control the evolution of the service contract. Given that backwards and forwards compatibility considerations are intentionally disregarded, you do not need to concern yourself with the impact of any change has on existing consumers

because all changes effectively break the contract. The new service version will only be available to new consumers while older consumers continue to use the previous version of the service and its associated WSDL and XML Schema artifacts.

By forcing a new namespace upon the contract with each change, this strategy has the disadvantage of requiring existing service consumers to adopt the new version of the contract to obtain access to additional new functionality. This situation is avoidable using alternate versioning strategies. Consumers can only interact with the service using the old contract, requiring the provider to offer multiple contracts to the service to maintain (if possible) continuity of operation.

This approach imposes an avoidable burden to the provider and its consumers.

### **Flexible Versioning (Backwards Compatibility)**

A common approach used to minimize the impact of change to a service's contract is to recognize backward-compatible changes that do not require a new contract namespace, and only version changes that are plainly incompatible.

This means that any backwards-compatible change is considered safe in that it extends or augments an established contract without affecting any of the service's existing consumers. A common example of this is adding a new operation to a WSDL definition or adding an optional element declaration to a message's schema definition. As with the Strict Versioning strategy, any change that breaks the existing contract results in a new contract version being required which is usually implemented by changing the target namespace of the WSDL definition and potentially also the XML Schema definition.

The primary advantage to this approach is that it can be used to handle a variety of changes while consistently retaining the contract's backwards compatibility. However, when compatible changes are made these become permanent and cannot be reversed without introducing an incompatible change.

Careful consideration is required to evaluate proposed changes such that contracts do not become overly bloated or convoluted. This is an especially important consideration for agnostic services that are heavily reused. Also, care must be taken not to abuse the flexible strategy by

creating compatible changes in a contract when the business requirements are incompatible. This might be done with the intention of trying to avoid the cost of creating a new version. The issue that arises is that the contracts then fail to validate messages. Furthermore, consumers can no longer use the contracts as basis for knowing the requirements and constructing proper messages accordingly.

### **Loose Versioning (Backwards and Forwards Compatibility)**

As with the previous two approaches, the Loose Versioning strategy requires that incompatible changes result in a new service contract version. The difference is in how service contracts are designed initially. Rather than strictly targeting schema and payload requirements known at design time, WSDL and XML Schema anyType and wildcard capabilities respectively can be employed to anticipate and accommodate future message payload changes. For example:

- The anyType attribute provided by WSDL allows a message to consist of any valid XML document
- XML Schema wildcards can be used to allow a range of unknown data to be passed in message definitions

Because wildcards allow undefined content to be passed through service contracts provides an opportunity to further expand the range of acceptable message element and data content. On the other hand the use of wildcards may result in vague and overly coarse service contracts that place the burden of validation on the underlying service logic.

## **Versioning Contracts**

There are many ways to version contracts. This document describes how to version these using target namespaces, file paths, service QNames, and documentation.

The advantage of leveraging target namespaces for versioning is that you can control which versions of the documents are referred to. For example you can designate that a v2 BPEL process refer to a v2 WSDL document and v2 schema definition. Target namespaces should be associated with a major version. If a new major version is required, the target namespace should be updated to signal this, for example: <http://myproject/v2>.

By simply changing the target namespace of a BPEL definition, you can have many versions of a *process plan* – a versioned unit of deployment in ActiveVOS - living in parallel on an ActiveVOS server. You cannot however deploy an incompatible change to the ActiveVOS Resource Catalog and in the process overwrite an existing schema or a WSDL artifact if it is being used by long running processes. Doing so will result in an incompatibility and will result in faulting a long running process when it executes next.

By placing different contract versions in different folders or file paths, you can have many versions of the contract documents in your orchestration project. Using such a file structure, each version of the contract documents can live simultaneously on a server (and resolved uniquely in the ActiveVOS Resource Catalog). For example, under the WSDL folder of a project, you should consider using sub-folders for each of the versions such as WSDL/v1, WSDL/v2, and so on.

Employing the use of a version-specific target namespace and file paths, will allow you to invoke the new service version whose QName service name in the Deployment Descriptor (PDD) is distinct. The PDD is where the actual endpoints (service addresses) are defined either explicitly or via indirection using a URN mapping. For example, BPEL process plan v2 should have a new service name (QName) like myplan\_v2.

Lastly, since target namespaces only document major versions, you should additionally document major and minor versions in your WSDLs and schemas for easy reading. Schema file versions can be documented by the version attribute. WSDLs can be versioned with the documentation element. There is an added benefit to documentation element as it can be read plainly in the service definitions of the ActiveVOS admin console.

## How to Version a Schema document

To version a schema document you must update the target namespace and the version attribute as follows.

- Minor and major contract version numbers are expressed using a version attribute of the schema element. You should update the version attribute in the header to declare major and minor versions such as version="2.0", but this has no functional meaning otherwise.

- Major version numbers are appended to the schema's target namespace and prefixed with a "v" such as shown here:

`targetNamespace=http://www.example.org/VersionProvider/v2`

The exception to the preceding rule is when the first version of a schema is released. In this case, no version number is added to the namespace. A compatible change increments the minor version number and does not change the schema's target namespace. An incompatible change increments the major version number and results in a new target namespace for the schema.

You must then update WSDLs to refer to the new namespace so that the WSDLs will point to the new schema version and not the old one. You must also update the namespaces referenced by BPEL process to point to the new schema target namespace. To do use ActiveVOS Designer's outline view to make the updates.

Different versions of a schema should be placed in unique folders to give them unique file paths. The file path makes each file version distinct from the other versions and allows the different versions to exist in parallel on the server. When referring to a new schema version, you must update your BPEL process imports using the Outline View of ActiveVOS Designer and updated them to point to the new schema file.

Here is an example of a versioned schema with changes in bold:

```
<Schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/VersionProvider/v2"
xmlns:tns="http://www.example.org/VersionProvider/v2"
elementFormDefault="qualified"
version="2">
```

## How to Version a WSDL document

To version a WSDL document you must change its target namespace in a way that declares the version or release. You should also document the version in the documentation element.

- Minor and major contract version numbers are expressed using a documentation element that follows the opening definitions element. The version numbers are displayed after the word "Version" as follows:

```
<documentation>Version 1.0</documentation>
```

An additional benefit is that this documentation is visible in the service definitions page of the ActiveVOS server. If you look at the WSDL of a service definition, you will see something like the following:

```
<wsdl:definitions
targetNamespace="http://VersionProvider/v1">
<wsdl:documentation>Version 1.0 </wsdl:documentation>
<p1nk:partnerLinkType name="VersionProviderPLT">
<p1nk:role name="Provider" portType="tns:VersionProvider"/>
</p1nk:partnerLinkType>
```

- Major version numbers are appended to the WSDL definition's target namespace and prefixed with a "v" as shown here:

```
targetNamespace=http://VersionProvider/v2
```

The exception to the preceding rule is when the first version of a WSDL definition is released. In this case, no version number is added to the namespace. A compatible change increments the minor version number and does not change the WSDL definition's target namespace. An incompatible change increments the major version number and results in a new target namespace for the WSDL definition.

Different versions of a WSDL should be placed in unique folders to give them unique file paths. The file path makes each file version distinct from the other versions and allows the different versions to exist in parallel on the server. When referring to a new WSDL version you must update your BPEL process imports in ActiveVOS Designer's Outline View and update these to point to the new WSDL file location.

Here is an example of a versioned WSDL with changes in bold:

```
<wsdl:definitions targetNamespace="http://VersionProvider/v2"
xmlns:tns="http://VersionProvider/v2"
xmlns:types1="http://www.example.org/VersionProvider/v2"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>Version 2.0</wsdl:documentation>
```

## Artifact Handling Best Practices

### Use of Relative Paths

It is a best practice for processes, WSDL and XML Schema document to use relative paths when importing artifacts. The primary reason to do so is to ensure that BPEL processes, WSDL and XML Schema imports are portable between environments and tooling.

Use of the ActiveVOS `'project:/'` Project Location URI is not required and recommended for use in XML Schema and WSDL imports. This URI is used by the Process Deployment Descriptor to identify the path to deployment artifacts based on their location relative to the root directory of the project. This location will be the location where artifacts are deployed to in the ActiveVOS Resource Catalog.

The Project Location URI is also used for ActiveVOS extension functions such as `getCatalogResource()` which requires a `'project:/'` type Project Location URI as a hint to locate the resource in the catalog.

### Use of a Common Repository Project for Shared Artifacts

Organizations often standardize and create shared schema to house and consolidate approved and reusable schema definitions in one location in a schema repository. They do so to avoid introducing incompatible definitions requiring unnecessary transformation for example. These teams also often take on the responsibility to manage namespaces so as to avoid conflicts.

When this is the case the best practice is to create an ActiveVOS project in the workspace to hold these shared artifacts. An example might be a common repository project (e.g. `'project:/repository'`) where a flat list or a tree of schema and WSDL definitions are defined. If the tree of schema is used, all imports need to use a relative path.

When importing schema from a shared project, a BPEL process in `'project:/ProjectA'` should use the relative path of `'../../../repository/xsd/schema123.xsd'` to import definitions of `'schema123.xsd'`. Versioned documents will of course have versioning information in the document's path as described previously.

## Understanding ActiveVOS Process Versioning and Revisions

ActiveVOS process versioning allows different versions for a given process to exist in ActiveVOS Server. Two deployments are considered to be different versions of the same process if they have the same target namespace and name in the BPEL file, but one deployment differs from the other in some way. This allows for minor revisions of a process to be deployed without forcing a major version change. However, when WSDL and Schema artifacts that are consumed by the process contain incompatible updates, deploying updates can have undesirable effects such as causing a running process to fault because it encountered an incompatible change in such artifacts.

ActiveVOS process versioning is a deployment feature that allows you to control when processes become effective and for how long. You can also control what happens to processes created by older versions when a new version becomes effective. While multiple versions of a process can exist concurrently, only the latest effective version is capable of creating new process instances.

The latest effective version is in a *current* state. Other states include *future*, to describe versions that have an effective date in the future, *expired* to describe versions whose expiration date has arrived or has been set, and *inactive* to describe expired versions that no longer have running process instances.

The process deployment descriptor provides selections for describing how a deployment is to be versioned. These selections are all optional and have default values as described below.

The following example shows the syntax for version information in the .pdd file.

```
<version effectiveDate="2005-12-12T00:00:00-05:00"
expirationDate="2007-12-12T00:00:00-05:00" id="1.5"
runningProcessDisposition="migrate"/>
```

where:

- **Effective date** is the date the new version becomes the current version and all new process instances run against it. Depending on the disposition selected for running processes, some may continue to run until complete using the older version.
- **Expiration date** is the date, beyond the effective date, the current version expires. An expired version is not capable of

creating new process instances. Once all of the running processes tied to an expired version complete then the version becomes inactive. All process instances for the current version run to completion.

- **Id** is the process version number in major.minor format. You do not need to provide a version number. ActiveVOS server auto-increments new versions. The server increments a version number by dropping the minor value and adding 1 to the max number. For example, version 1.5 increments to version 2.0.
- **Running process disposition** is the action the server takes on any other versions of the same process that currently have processes executing once this version's effective date arrives.

Valid values for the running process disposition parameter are:

- **Maintain.** Indicates that all process instances for the previous versions should run to completion. This is the default value.
- **Terminate.** Indicates that all process instances running under previous versions should terminate on the effective date of the new version, regardless of whether or not the process instances are complete.
- **Migrate.** All running process instances created by previous versions will have their state information migrated to use the newly deployed process definition once its effective date arrives. If there are incompatible changes between the versions that would not permit them to be migrated, you receive an error message during deployment of the new process, and its deployment fails. At the time of this writing, allowable changes are limited to expression language changes in the process.

## ActiveVOS Deployment Considerations

### Criteria for Deploying a Process Revision

A revision to a deployed process must meet the following criteria:

- The fully qualified process name must not change
- A revision can include a change to either the BPEL definition or the process deployment descriptor
- A revision is not new if the auto-increment deployment feature determines that the BPEL definition and the deployment

descriptor are the same as the deployed version having the highest version number. If the BPEL definition and the descriptor file do not contain any differences, then the version in the ActiveVOS server's catalog is up-to-date and no deployment occurs.

- If you need to modify existing WSDL-related properties, such as partner links or correlation properties, then you should create a new process, not a new revision

### **Use of Common Business Process Archives**

A business process archive (BPR) can contain all artifacts that are required to support the execution of a process. A BPEL process may also import resources from a resolvable HTTP address, and in such cases it isn't necessary to include the resource in the BPR as it will be looked up at runtime.

In the case of shared resources such as schema and WSDL definitions, a BPR can be created to deploy these artifacts to the ActiveVOS Resource Catalog. This archive and the ActiveVOS project that contains the artifacts should be controlled by a team. Changes and deployment of this archive needs to be managed such as to avoid introducing non-backwards compatible updates that would negatively impact previously deployed processes that import its definitions.

Although it is possible for a single BPR archive to deploy common resources such as all WSDL definitions in one deployment there is a limitation in ActiveVOS 7: the public and private WSDLs that are generated by ActiveVOS Designer must be deployed in the same archive used to deploy the process definition that imports these. More specifically, any process that directly imports a public/private WSDL needs the WSDL in its BPR deployment archive. As such this does not apply only to the process that owns the public/private WSDL.

### **Avoiding the Deployment of WSDL Documents of the Same Namespace and WSDL Service QName to the ActiveVOS Resource Catalog**

ActiveVOS allows the ability to create copies of an artifact in the Resource Catalog that share the same namespace but are found at different locations because they originate from different ActiveVOS projects. While it is permissible to do so, these artifacts can evolve

independently in their respective projects yet share the same namespace, but special care needs to be taken for WSDL artifacts.

Where conflict arises and becomes problematic is when a service being invoked shares a WSDL of the same namespace and WSDL service QName of another service. At runtime ActiveVOS looks up WSDLs by namespace and matches the service to invoke by WSDL service QName when performing an invoke operation. At all other times ActiveVOS looks up artifacts based on the associated deployment plan.

What specifically must be avoided is an overlap of the service QName definition within a WSDL with others. Issues arise when for example two different versions of a WSDL document get deployed to the Resource Catalog each with the same namespace, and yet each defining an incompatible version of the `tns:MyWSDLService` service. At runtime this results in ambiguity when ActiveVOS looks up the service by WSDL Service QName. This gives rise to the common runtime “operation not found for PortType” error being reported because two or more developers have deployed WSDL documents sharing the same namespace with conflicting service QNames, each exposing entirely different operations.

If deploying multiple WSDL in the same namespace, care needs to be taken to define service definitions that do not have overlapping service QNames. Using separate target namespaces (as part of your interface/service versioning strategy) will help avoid this situation.

### Relative Paths for Portability

Using relative path imports allows developers to move projects in an ActiveVOS Workspace between machines and environment, and ensures that imports still resolve. This applies both at design time and also at runtime. This also applies to supporting your versioning strategy.

For example if a schema at location hint

`'project:/myproj/xsd/schema1.xsd'` in the resource catalog (and a developer's workspace) has an import that is relative such

`'../../..../repository/xsd/schema2.xsd'`, then at runtime ActiveVOS will resolve the import to location hint

`'project:/repository/xsd/schema2.xsd'` that had been specified by the process deployment descriptor used to deploy `'project:/common'` artifacts.

## Governing Contracts

Understanding the techniques described in this document are required but not sufficient. There is also the challenge of managing contract versioning in a large system or organization. Development organization need to develop ways (i.e. processes) for reviewing contract changes, determining what versioning strategy to use, implementing the versioning strategy, and making sure that all affected parties refactor their processes accordingly.

Organizations should consider the establishment of a contract governance process and role. Two broad processes are suggested here – the merits of which are left to the reader’s needs.

### Proactive Governance

1. A “Contract Master” role is responsible and owns all WSDL and schemas and has exclusive rights to update and deploy. This requires all projects be prohibited from deploying imported WSDLs and Schemas. During deployment the PDD needs to be configured by un-selecting the option to replace existing resources in the export dialog when creating a new BPR archive.
2. Developers must petition for a change in a contract
3. The “Contract Master” then assesses the change and determines if a new contract version is required
4. If a new contract version is created, all affected developers are notified
5. Developers must then refactor their processes to incorporate the new contract version
6. Test teams are notified of the contract change

### Reactive Governance

1. Developers make contract changes during development
2. Before deployment, the “Contract Master” calls a freeze to contract changes (trusting that developers know how to avoid inadvertent changes).
3. “Contract Master” reviews contract documents and determines where new contract versions are required
4. “Contract Master” creates new contract versions
5. Developers must refactor their processes to incorporate the new contract version
6. Test teams are notified of the contract change

Define the process that suits you best. But please be assured that unconstrained updates to the ActiveVOS Resource Catalog with incompatible contracts documents will result in wasted efforts isolating the errors that will result.

## Versioning ActiveVOS Projects

### Refactoring Service Provider Projects

The service provider's ActiveVOS Designer project must be refactored if it publishes a new contract version.

- New folders should be created with the respective version names and should contain the new versions of the BPEL, WSDL, and schemas.
- New target namespace for new versions of BPEL process to allow the new BPEL plan to live in parallel to the old plan version.
- Namespace references in the outline must be updated to point to the new WSDL and schema versions.
- Imports in the outline must be updated to point to the new WSDL and schema versions.
- If you are using public WSDLs, you must open the file and update the imports to point to the new WSDL version.
- Assigns must be refactored to take into account any changes in the message variables.
- PDD of the new BPEL version must have new MyRole service names (QName) which are distinct from the service QName of the old version.

### Refactoring Service Consumer Projects

The consumer's ActiveVOS Designer project must be refactored if a service provider publishes a new contract version.

- Namespace references in the outline must be updated to point to the new WSDL and schema versions.
- Imports in the outline must be updated to point to the new WSDL and schema versions. This is done by editing the provider

imports to point to the new versions and refreshing the imports node in the outline.

- Variables will automatically be updated
- Assigns must be refactored to take into account any changes in the message variables.
- In the PDD, you must edit the PartnerRole to point to the new version. This is done by clicking on the partner link of the service provider you need to update, clicking on the ellipse, selecting the project of the service provider, and drilling down to select the service name of the new version.

### Versioning POJO projects

Since a plain old java object (POJO) project calls the POJO as a service provider behind the scene, changes in the POJO should not concern service consumers unless the POJO project needs to change its public-facing interface. When a POJO project needs to be versioned, you may follow the usual steps for the BPEL, WSDL, and schema which are part of the service contract with the consumers. The POJO classes can be versioned using a flexible strategy by adding new methods along side of the old methods. Both BPEL versions can use the same POJO. The old methods can then eventually be deprecated.

There are a couple of approaches to versioning POJOs that would have you change either 1) the classname or 2) the project location of the jar (or both if you want). The Java Service invoke handler parameter is illustrative of this. E.g.

```
com.active.classiccars.rules.CarRules?inherit=true&classpath=project:/ActiveVOSDemoRules/classicCarsRules.jar
```

The combination of the classname and the project location are what makes a given version of a pojo to be loaded.

You also need to be careful of POJO dependencies. If a POJO depends on any other jars in the server's classpath there could be versioning issues if any of these jars change.

## About Active Endpoints

Active Endpoints' ([www.activevos.com](http://www.activevos.com)) ActiveVOS is the business process management system (**BPMS**) that development teams will love. ActiveVOS empowers project teams to create business process management (**BPM**) applications using services, making their businesses more agile and effective. ActiveVOS promotes mass adoption of SOA-enabled BPM applications by focusing on accelerating project delivery time with a complete, affordable and easy-to-use system. Active Endpoints is headquartered in Waltham, MA with development facilities in Shelton, CT.

To find out how Active Endpoints can help your business, visit <http://www.activevos.com>, call +1 781 547 2900 and press 1 for Sales, or email us at [info@activevos.com](mailto:info@activevos.com).