SAP NetWeaver
How-To Guide

How To... Fast-Switch Integration Scenarios Between SAP PI Runtimes Part I: Directory API

Applicable Releases:

SAP NetWeaver Process Integration 7.1
(Including Enhancement Package 1)

Topic Area:

SOA Middleware

Capability:

Service Bus

Version 1.0

July 2010

Document History

Document Version	Description
1.00	First official release of this guide

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

1. Scenario

You plan to deploy additional SAP NetWeaver Process Integration (PI) runtime engines – e.g. Advanced Adapter Engine (AAE) or full PI instance – in your PI system landscape and you want to be able to quickly switch the designation of the adapter engine runtime and/or route application sender/outbound requests to the proper PI runtime for many or all your AAE local processing integration scenarios.

For example, in potential SAP NetWeaver PI federation scenarios, additional PI runtime engines (e.g. Advanced Adapter Engine) can be deployed to support business continuity in planned downtime scenarios. Scenarios can be kept up and running by switching from one runtime to another before the downtime takes place and then switched back once the system is brought back up. As another example, a new Advanced Adapter Engine can be stood up to alleviate excessive resource usage on the central Advanced Adapter Engine. You can quickly configure channels to point to the new AAE instance upon deployment and also quickly route appropriate sender application request to the new adapter engine.

This How-to Guide is the first of a two-part series providing details of how to fast-switch integration scenarios from one SAP PI runtime to another. This part focuses on how the *Integration Directory Programming Interface* (Directory API) can be used to quickly modify communication channels in mass such that the Advanced Adapter Engine setting can be changed to a different Advanced Adapter Engine in the SAP NetWeaver PI landscape.

2. Background Information

In SAP NetWeaver PI, adapters are used to enable communication to various types of system entities (e.g. web service consumers/provides, file systems, databases, JMS providers, etc.). Adapters reside in the Adapter Engine. Configuration of adapters takes place in the Integration Directory using *Communication Channel* objects and each channel must designate which Adapter Engine to use at runtime. At a given customer site, there could be anywhere between tens to hundreds (or more) of channels created and configured. To manually change the settings for a large number of channels could prove to be an arduous task. This is where the Directory API comes in.

The *Integration Directory Programming Interface* (Directory API) can be used to access, edit, and activate objects within the Integration Directory (e.g. communication channels) in mass. The programming interface consists of a set of web services that are fully implemented on the Java application server of the PI instance. Using WSDLs, any web service client can be used to call the Directory API web services. Leveraging the Directory API, channels can be configured in mass to quickly switch the Adapter Engine designation from one to another.

3. Prerequisites

The following prerequisites should be considered:

- Working knowledge of SAP NetWeaver PI and introductory knowledge of Directory API.

  Note

  This guide is not intended to provide a detailed introduction to the Directory API, but rather how specific Directory API services can be used to facilitate the process of performing a mass switch from one Adapter Engine to another. For more information on the Directory API, proceed to the SAP Help Portal: Integration Directory Programming Interface.

Note

As of SAP NetWeaver PI 7.1 EHP1, there are two versions of the Directory API – an earlier version that was delivered with releases 7.1 (and lower) and a new version delivered starting with 7.1 EHP1 (for compatibility reasons, the earlier version is also is still available in 7.1 EHP1). The main difference between the two is that new services are available in the new Directory API version that accommodates objects delivered starting from release 7.1 (e.g. Direct Connection).

Important

Though with release 7.1 many new configuration objects were introduced, only the earlier version of the Directory API is available for this release.

- SAP NetWeaver XI/PI release requirements:
  - To be able to use the earlier version of the Directory API in the context of fast-switching AAE local processing scenarios, the following release levels must be met: SAP NetWeaver 7.1
  - To be able to use the new version of the Directory API, the following release level must be met: SAP NetWeaver 7.1 EHP1 (earlier version also available for compatibility reasons).
- User roles:
  - **SAP_XI_API_DEVELOP_J2EE** (for any operation, particularly Change, Create, OpenForEdit, Revert, Delete, CreateFromTemplate, Activate, Revert)
  - **SAP_XI_API_DISPLAY_J2EE** (for the operations Query, Read, Check, GetState, CheckContent, GetCacheState, GetObjectIdentifier)

  Note

  These are Java engine roles that you can administer using URL: http://<host>:<J2EE-Port>/useradmin.
- At least one decentral adapter engine is installed and configured for use with the PI system.
- User Interface development experience.

  Note

  Although not strictly required to execute the required Directory API web services, an appropriate user interface (UI) should be developed in order to simplify the overall task of switching the Adapter Engine channel configurations and making it more user friendly. As such, UI development knowledge or availability of a resource with such knowledge is recommended. Which UI technology chosen is not so important so long as web service calls can be incorporated into the UI development process.

4. Step-by-Step Procedure

Setting the *Adapter Engine* parameter in a channel configuration is normally done through the standard Integration Directory user interface. If you open up any Communication Channel object, you can see this parameter.

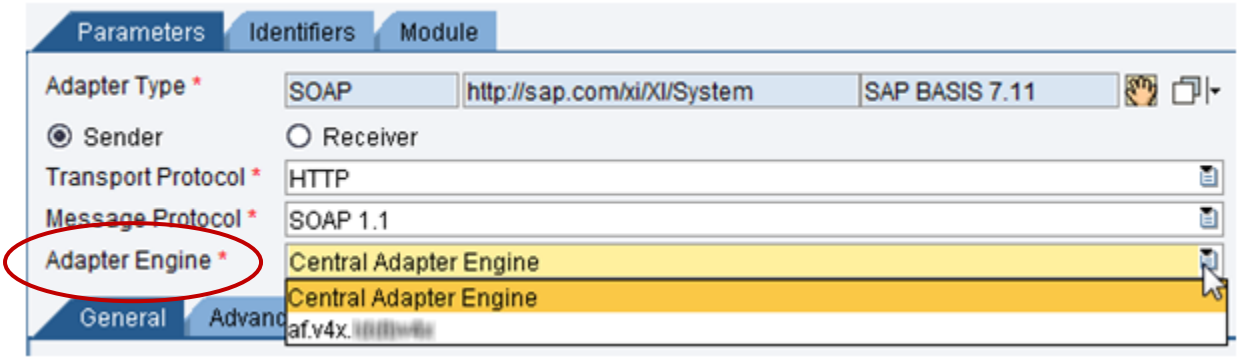


Figure 1: Typical Communication Channel *Parameters* tab where *Adapter Engine* is configured

As shown in Figure 1, no matter which channel type (e.g. SOAP, JDBC, File, etc.) is selected, the Communication Channel *Parameters* tab is where the *Adapter Engine* setting is configured. All available adapter engines that are currently configured for a PI system will be displayed in the drop down window. In the above case, there is one other adapter engine ("af.v4x.<server>") configured along with the central adapter engine for this PI system.

The core idea of the adapter engine fast switching process is to easily switch this Adapter Engine setting from one to another for any specified number of channels in mass using the Directory API.

Below is a more detailed description of the process of how to use the Directory API services to enable fast switching of the Adapter Engine configuration.

4.1 Choose a UI Technology

As mentioned earlier, the development of a user interface (UI) is highly recommended as several web service operations are required to be called in succession to perform the fast switching process. Which UI technology chosen is not so important as long as it can easily incorporate web service calls during into the UI development. Examples of UI technologies that can be used include SAP Web Dynpro (ABAP or Java). Other non-SAP UI technologies are also fine as the Directory API fully complies with web service standards and there is no dependency to a specific SAP technology.

Note

How the UI is arranged and developed is completely up to the customer. In the sections that follow, some suggestions are provided for UI components to adequately perform the adapter engine fast switch functionality. These suggestions are to be taken as a minimum guideline and customers should feel free to extend and embellish their UI as it suits their needs.

4.2 Obtain WSDLs for Required Directory API Services

In order to enable fast switching of the adapter engine, two Directory API services are required.

1. **Communication Channel** service
2. **Change List** service

Table 1 below lists the specific Directory API services required according to the SAP NetWeaver PI release level.

XI/PI Release	Interface Name	Service Name	Namespace
7.1 EHP1	CommunicationChannelIn	CommunicationChannelInService	http://sap.com/xi/BASIS
7.1 EHP1	ChangeListIn	ChangeListInService	http://sap.com/xi/BASIS
7.1 and below	CommunicationChannelServiceVi	CommunicationChannelService	urn:CommunicationChannelServiceWsd/CommunicationChannelServiceVi
7.1 and below	ChangeListServiceVi	ChangeListService	urn:ChangeListServiceWsd/ChangeListServiceVi

Table 1: Directory API services required for fast switching process

The WSDLs for the Directory API services can be obtained from two main sources:

1. Enterprise Services Repository (ES Repository)
2. Web Service Navigator (WS Navigator)

4.2.1 ES Repository as WSDL Source

The WSDLs for the Directory API services are delivered in the *SAP BASIS* software component of the Enterprise Services Repository (ES Repository). The exact location of the WSDLs will differ slightly depending on the SAP NetWeaver PI release. The ES Repository can be accessed using one of the following URLs: *http://<pi host>:<j2ee port>/dir* (release 7.1 and higher).

Note

As the ES Repository is a design time resource, the WSDLs obtained from it will not contain system endpoint details (e.g. binding for HTTP access and the service port). These missing pieces must be completed before the Directory API services are called at runtime.

4.2.1.1 Release 7.1 EHP1

1. Log on and open the ES Repository and navigate to the software component version *SAP BASIS 7.11 →http//sap.com/xi/BASIS →Integration Directory API*.
2. For the Communication Channel service, navigate to folder *Communication Channel →CommunicationChannelIn* and open the *CommunicationChannelIn* service interface.

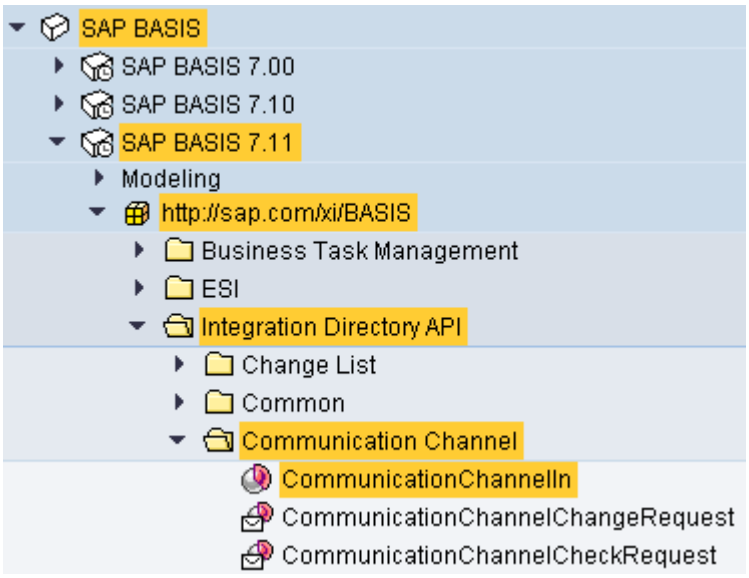


Figure 2: ES Repository – location of service interface *CommunicationChannelIn* in release 7.1 EHP1

3. Proceed to the *WSDL* tab on the service interface page and export the WSDL using the *Export WSDL to File* button. Store the WSDL locally where it can be accessed later.

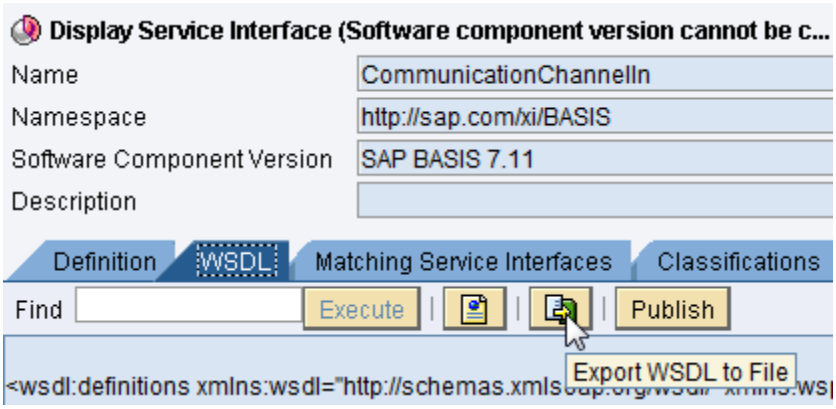


Figure 3: Export WSDL from service interface (releases 7.1 EHP1 and higher)

4. Repeat the above steps to obtain the WSDL for the Change List service.

4.2.1.2 Releases 7.1 and Below

1. Log on and open the ES Repository and navigate to the software component *SAP BASIS* and open the *CommunicationChannelService* external definition.
   - Go to *SAP BASIS 7.10→http://sap.com/xi/XI/System→External Definition→CommunicationChannelService*.
2. Export the WSDL by going to the *Tools* menu and selecting *Export Original Document*. Store the WSDL locally where it can be accessed later.

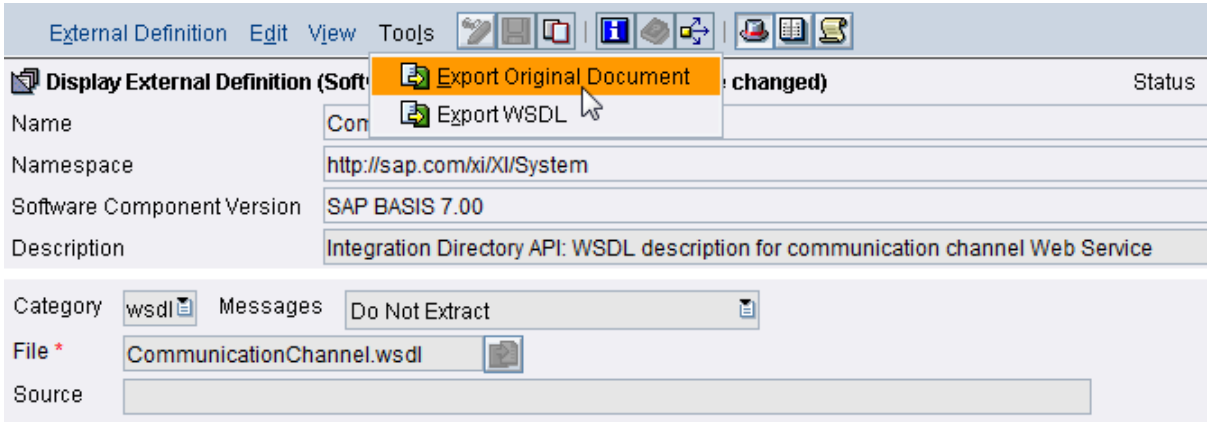


Figure 4: Export WSDL from external definition (releases 7.1 and lower)

3. Repeat the above steps to obtain the WSDL for the Change List service.

4.2.2 WS Navigator as WSDL Source

Another way to obtain the WSDLs is by using WS Navigator. By using WS Navigator of the particular SAP NetWeaver PI system, you can get WSDLs that include specific endpoints (e.g. bindings for HTTP access and service ports) for that PI system. The WS Navigator can be accessed using the following URL: *http://<pi host>:<j2ee port>/wsnavigator*.

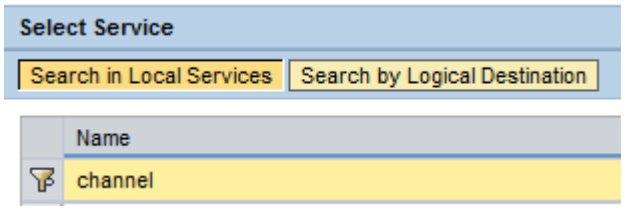
4.2.2.1 Release 7.1 EHP1

1. Log on and open the WS Navigator on your SAP NetWeaver PI system. If the *Service Information* tray is not expanded, do so by clicking on the button on the far right.
2. Select *Provider System* radio button for *Search Type* and enter “channel” for the *Search for* value.
3. Select service *CommunicationChannelIn*. The Service Information tray should now be populated with data.
4. Obtain the WSDL by clicking on the *WSDL URL*.
5. Repeat the above steps to obtain the WSDL for the *ChangeListIn* service. Use “change” for the search value.

4.2.2.2 Release 7.1

1. Log on and open the WS Navigator on your SAP NetWeaver PI system.

2. Enter “channel” as the filter criteria and click the filter button.



3. Select the resulting *…CommunicationChannelServiceVi* service.
4. Obtain the WSDL from the *WSDL URL* link provided.
5. Repeat the above steps to obtain the WSDL for the ChangeListIn service. Use “change” as the filter criteria.

4.3 How to Use the Services

Once the WSDLs are obtained, web service client applications can import them and generate the necessary code to execute and process the web service calls. For the adapter engine fast switching process, only two services are required: *Communication Channel* and *Change List*. However, multiple operations are required from each service and, for maximum efficiency, they should be called in a particular sequence. In this section, we examine the overall call sequence and look at the details of the service/operations used.

4.3.1 Service Call Sequence Overview

Though the *Communication Channel* and *Change List* services contain multiple operations (as listed below), only a few operations are required for the adapter engine fast switching process. Those operations are *Query*, *Read*, and *Change* for the *Communication Channel* service and *Create* and *Activate* for the *Change List* service.

Communication Channel	Change List
Change	**Activate**
Check	CheckContent
Create	**Create**
CreateFromTemplate	GetCacheState
Delete	GetObjectIdentifiers
OpenForEdit	GetState
Query	Revert
Read	
Revert	

Table 2: Operations list for required services

In general, the recommended service-operation call sequence looks as shown below in Figure 5. In the next section, we take a closer look at each of the operations used and their respective roles in the adapter engine fast switching process.

Figure 5: Service-operation call sequence overview

4.3.2 Service Processing Details

At a minimum, the services and operations specified in Figure 5 must be used. In this section we look at why these operations are necessary and the details of how they should be processed. We will also show what the structure of the request and response look like for each of the operations.

Note

The request and response structure screenshots are taken from the ES Repository of a SAP NetWeaver PI 7.1 EHP1 system. Although the name and types may not be an exact match with lower releases, they are similar enough to easily deduce the correspondence of the structures between releases.

4.3.2.1 Change List – Create

The first service operation that should be called is the *Change List – Create* operation. This is because there are no default change lists (e.g. with the name "Standard Change List") when using the Directory API, which means that you must explicitly specify a change list on your own. Otherwise a new change list name will be automatically generated each time based on a combination of username and time stamp (e.g. “Created with API on Jan 1, 2010 12:00:00 PM by user <userid>”). If your intention is to create or change many objects at once (as in our fast switching case), then this can cause an unmanageable number of change lists to be generated quickly.

To keep all newly changed communication channel objects in a single change list, it is recommended to explicitly create a change list as the first step in the call sequence using the *Change List Create* operation.

Request

Name	Category	Type	Occurrence
ChangeListCreateRequest	Element	RestrictedChangeListID	
Name	Element	p1:LANGUAGEINDEPENDENT_	0..1
Description	Element	p1:LONG_Description	0..1
languageCode	Attribute	p1:LanguageCode	optional

Response

Name	Category	Type	Occurrence
ChangeListCreateResponse	Element	ChangeListCreateOut	
ChangeListID	Element	ChangeListID	0..1
ChangeListID	Element	ChangeListChangeListID	0..1
Name	Element	p1:LANGUAGEINDEPENDENT_	0..1
Description	Element	p1:LONG_Description	0..1
languageCode	Attribute	p1:LanguageCode	optional
LogMessageCollection	Element	LogMessageCollection	0..1

Key Notes

- First web service operation that should be called for the adapter engine fast switch process.
- All elements in the request are optional.
- If desired, an appropriate, arbitrary name and description can be specified in the request.
- If Name is not specified, the service operation will generate the name using a combination of username and time stamp (e.g. “Created with API on Jan 1, 2010 12:00:00 PM by user <userid>”).
- The response provides the *ChangeListID* which is what needs to be used every time a *Communication Channel – Change* request is executed.
- Using the Directory API you can only edit your own change lists. You cannot take on and edit the change lists of other users.
- UI should ideally include a user input field for the Change List *Name*. Input field for *Description* is “nice to have”.

4.3.2.2 Communication Channel – Query

The *Communication Channel – Query* operation is essentially a search operation. In order to perform changes in mass to communication channel objects, this service operation is required as the resulting list of communication channels in the response can serve as input for the next service operation *Communication Channel – Read*.

Request

Name	Category	Type	Occurrence
CommunicationChannelQueryRequest	Element	CommunicationChannelQueryIn	
CommunicationChannelID	Element	CommunicationChannelID	0..1
PartyID	Element	CommunicationPartyID	0..1
ComponentID	Element	CommunicationComponentCom	1
ChannelID	Element	CommunicationChannelChannel	1
Description	Element	p1:LONG_Description	0..1
languageCode	Attribute	p1:LanguageCode	optional
AdministrativeData	Element	ObjectAdministrativeData	0..1
ResponsibleUserAccountID	Element	UserID	0..1
LastChangeUserAccountID	Element	UserID	0..1
LastChangeDateTime	Element	p1:GLOBAL_DateTime	0..1
FolderPathID	Element	p1:LANGUAGEINDEPENDENT_T	0..1

Response

Name	Category	Type	Occurrence
CommunicationChannelQueryResponse	Element	CommunicationChannelQueryOut	
CommunicationChannelID	Element	CommunicationChannelID	0..unbounded
PartyID	Element	CommunicationPartyID	0..1
ComponentID	Element	CommunicationComponentComponentID	1
ChannelID	Element	CommunicationChannelChannelID	1
LogMessageCollection	Element	LogMessageCollection	0..1

Key Notes

- Essentially a search operation.
- Can search by communication channel elements (e.g. party, component, channel), description, or administrative data (e.g. user ID).
- All elements are optional. If the request is empty, all communication channels are returned.
- Wildcard characters such as "*" can be used in each field/element.
- Communication channels returned in the response serve as input for next service operation, *Communication Channel – Read*.
- UI should include user input fields for *PartyID*, *ComponentID*, and *ChannelID* at a minimum. "Nice to haves" are input fields for *Description* and user ID related fields.
- UI should also include a "Search" related button that triggers the execution of the *Query* and *Read* operation in succession – for reason, see next section.

4.3.2.3 Communication Channel – Read

One might ask why a *Read* operation is necessary if a *Query* is already being executed or perhaps what the difference is between the *Read* and *Query* operation.

Although the request of both operations use the *CommunicationChannelID* element, the *Read* operation requires an exact match in the input for *PartyID*, *ComponentID*, and *ChannelID* in order for a resulting channel to be returned (e.g. no wildcards can be used). The cardinality of the *CommunicationChannelID* element for a *Read* is *0..unbounded* whereas for a Query its *0..1*. That means a *CommunicationChannelID* element must be created for every communication channel object sought. This is why a *Query* operation is necessary first. The response from the *Query* is used as input for the *Read* request.

The *Read* is necessary because only in the *Read* response is all the details for a channel object returned. The details are required when executing a *Communication Channel – Change* operation as described later.

Request

Name	Category	Type	Occurrence	Default	Details
CommunicationChannelReadRequest	Element	CommunicationChannelReadIn			
ReadContext	Element	ReadContextCode	0..1	User	enumeration="User, Active"
CommunicationChannelID	Element	CommunicationChannelID	0..unbounded		
PartyID	Element	CommunicationPartyID	0..1		maxLength="60"
ComponentID	Element	CommunicationComponentComponentID	1		maxLength="60"
ChannelID	Element	CommunicationChannelChannelID	1		maxLength="60"

Response

Name	Category	Type	Occurrence
CommunicationChannelReadResponse	Element	CommunicationChannelReadOut	
CommunicationChannel	Element	CommunicationChannel	0..unbounded
MasterLanguage	Element	p1:LanguageCode	1
AdministrativeData	Element	ObjectAdministrativeData	0..1
Description	Element	p1:LONG_Description	0..unbounded
CommunicationChannelID	Element	CommunicationChannelID	1
AdapterMetadata	Element	DesignObjectID	0..1
Direction	Element	CommunicationChannelDirection	1
TransportProtocol	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
TransportProtocolVersion	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
MessageProtocol	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
MessageProtocolVersion	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
AdapterEngineName	Element	AdapterEngineName	0..1
AdapterSpecificAttribute	Element	GenericProperty	0..unbounded
AdapterSpecificTableAttribute	Element	GenericPropertyTable	0..unbounded
ModuleProcess	Element	ModuleProcess	0..1
SenderIdentifier	Element	ChannelAdditionalIdentifier	0..1
ReceiverIdentifier	Element	ChannelAdditionalIdentifier	0..1
LogMessageCollection	Element	LogMessageCollection	0..1

Key Notes

- Communication channel elements (party, component, channel) require an exact match in order for communication channel object details to be returned. No wildcards allowed.
- Cardinality of the *CommunicationChannelID* element is *0..unbounded*. One *CommunicationChannelID* required for each communication channel object sought.

- *Query* response serves as input for *Read* request.
- *ReadContext* element only allows use of predefined values
    - "**User**" (default): Returns latest version (active or in process) of communication channel object. Value "User" is recommended for the adapter engine fast switch process.
    - "**Active**": Returns only latest active version of communication channel object
- *Read* response has all the details of a given communication channel object and serves as input for the next operation *Communication Channel – Change* request.
- *Password* attribute values are not returned in the Read response (more details in "Password Handling" in the next section.
- An empty (blank) value returned for *AdapterEngineName* corresponds to the default *Central Adapter Engine* (i.e. the Central Adapter Engine is currently configured for the given channel).
- UI should ideally include a mutli-selectable table displaying the key response elements from the *Read* response. Key response elements include *AdapterEngineName*, *PartyID*, *ComponentID*, *ChannelID*, *Name* (i.e. adapter type name such as "SOAP", "File", or "JDBC"), and *Direction*. Of course more elements can be included in the table display if so desired.
- The *Query* and *Read* operations should be executed in succession upon initiation of a communication channel "Search" from the user since the resulting channel details from the *Read* response should be displayed back to the user in the UI as a multi-selectable table as described above (not the *Query* response).

4.3.2.4 Communication Channel – Change

A key point to keep in mind when using the Directory API is that **there is no *delta* handling of attribute changes**. This means that when perform a *Change* operation, to modify an object, you will still need to specify all the attributes for that object, just as though it is a *Create* operation. This is the primary reason why a full *Read* response must be used as the input in the *Change* request even though we are only changing one communication channel attribute, specifically the *AdapterEngineName*. One exception to this no delta handling rule is for *password* attributes and only for the new version of the Directory API (i.e. 7.1 EHP1 and above). See below for more details.

In the context of the AAE fast switching functionality, upon successful execution of the *Communication Change – Change* operation, all channel objects "selected" for change will be modified with the new *AdapterEngineName* and placed in the specified change list.

Password Handling

For both the new and old versions of the Directory API, there is special handling of *password* attributes:

- **Earlier Version (7.1)**
    - Password attributes are not returned in the *Read* response and must be supplied in the *Change* request in order for the password to be set in the channel.
- **New Version (7.1 EHP1 and above)**

- Password attributes are not returned in the *Read* response. But there is *delta* handling for password attributes in the *Change* request. This means the Directory API will ensure that existing passwords are kept even if the *Change* request does not supply any password values.

Request

Name	Category	Type	Occurrence
CommunicationChannelChangeRequest	Element	CommunicationChannelCreateChangeIn	
ChangeListID	Element	ChangeListChangeListID	0..1
CommunicationChannel	Element	RestrictedCommunicationChannel	0..unbounded
MasterLanguage	Element	p1:LanguageCode	1
AdministrativeData	Element	RestrictedObjectAdministrativeData	0..1
Description	Element	p1:LONG_Description	0..unbounded
CommunicationChannelID	Element	CommunicationChannelID	1
AdapterMetadata	Element	DesignObjectID	1
Direction	Element	CommunicationChannelDirection	1
TransportProtocol	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
TransportProtocolVersion	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
MessageProtocol	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
MessageProtocolVersion	Element	p1:LANGUAGEINDEPENDENT_SHORT_Name	0..1
AdapterEngineName	Element	AdapterEngineName	0..1
AdapterSpecificAttribute	Element	GenericProperty	0..unbounded
AdapterSpecificTableAttribute	Element	GenericPropertyTable	0..unbounded
ModuleProcess	Element	ModuleProcess	0..1
SenderIdentifier	Element	ChannelAdditionalIdentifier	0..1
ReceiverIdentifier	Element	ChannelAdditionalIdentifier	0..1

Response

Name	Category	Type	Occurrence
ConfigurationObjectModifyResponse	Element	ConfigurationObjectModifyOut	
ChangeListID	Element	ChangeListID	0..1
ChangeListID	Element	ChangeListChangeListID	0..1
Name	Element	p1:LANGUAGEINDEPENDENT_LONG_Name	0..1
Description	Element	p1:LONG_Description	0..1
LogMessageCollection	Element	LogMessageCollection	0..1

Key Notes

- No ***delta*** handling of attribute changes. When performing the *Change* operation, all attributes for the communication channel object must be included in the *Change* request, just as though it is a *Create* operation.

  CAUTION

  If some attributes are missing when executing the *Change* operation, corresponding existing attributes for the communication channel accessed will be wiped out.

- **Complete** *Read* response serves as input for the *Change* request.
- **Passwords (earlier API version): Channels with existing passwords must supply password value(s) in the *Change* request.**
- **Passwords (new API version): Delta handling exists for passwords.**

- The *ChangeListID* returned from the *Change List – Create* operation should be used.
- Especially for the *Change* operation, the *LogMessageCollection* element should be checked for errors. See section on *LogMessageCollection Element* for further information.
- UI should include an input field (or drop down) for the *new* adapter engine name and a "Submit" related button to trigger the execution of the *Change* operation.

To change the Adapter Engine to "Central Adapter Engine", either a blank value or value in the form "af.<sid>.<database server>" should be used for the *AdapterEngineName* field.

4.3.2.5 Change List – Activate

The *Change List – Activate* service operation will activate all objects associated with the provided *ChangeListID*. In our case, the *ChangeListID* returned from the initial *Change List – Create* operation should be used.

It is recommended to make the execution of the *Activate* operation an optional step (e.g. using a checkbox in the UI to enable activation) so that the user can have the option to go the Integration Directory and examine the communication objects changed in the change list and confirm that the objects were modified according to expectations.

Request

Name	Category	Type	Occurrence
ChangeListActivateRequest	Element	ChangeListChangeListID	

Response

Name	Category	Type	Occurrence
▸ LogMessageCollection	Element	LogMessageCollection	

Key Notes

- *ChangeListID* from initial Change List – Create operation should be used.
- Successful *Activate* operation call returns response with empty *LogMessageCollection* element.
- Activation should be optional.
- UI should include a checkbox (or similar) to give an option to activate upon submission or not.

4.3.2.6 LogMessageCollection Element

If an error occurs for any Directory API service operation, then the service response displays an error message. You can find the structure of the error message in the *LogMessageCollection* error message. In this structure individual messages are displayed according to object types.

When an error occurs, the *LogMessageCollection* structure is always displayed completely. If no message was displayed for a particular object then this part of the structure remains empty.

Name	Category	Type	Occurrence
LogMessageCollection	Complex Type		
LogMessage	Element	LogMessage	0..unbounded
LogMessageChangeList	Element	LogMessageChangeList	0..unbounded
LogMessageParty	Element	LogMessageCommunicationParty	0..unbounded
LogMessageProcessComponent	Element	LogMessageProcessComponent	0..unbounded
LogMessageBusinessSystem	Element	LogMessageCommunicationComponent	0..unbounded
LogMessageBusinessComponent	Element	LogMessageCommunicationComponent	0..unbounded
LogMessageIntegrationProcess	Element	LogMessageCommunicationComponent	0..unbounded
LogMessageCommunicationChannel	Element	LogMessageCommunicationChannel	0..unbounded
LogMessageSenderAgreement	Element	LogMessageMessageHeader	0..unbounded
LogMessageReceiverAgreement	Element	LogMessageMessageHeader	0..unbounded
LogMessageDirectConnection	Element	LogMessageMessageHeader	0..unbounded
LogMessageIntegratedConfiguration	Element	LogMessageMessageHeader	0..unbounded
LogMessageReceiverDetermination	Element	LogMessageMessageHeader	0..unbounded
LogMessageReceiverRule	Element	LogMessageReceiverRule	0..unbounded
LogMessageInterfaceDetermination	Element	LogMessageMessageHeader	0..unbounded
LogMessageValueMapping	Element	LogMessageValueMapping	0..unbounded
LogMessageConfigurationScenario	Element	LogMessageConfigurationScenario	0..unbounded

So if an error occurs during execution of an operation for the *Communication Channel* service, you can find the error message along with the severity code, classification code, and information to identify the problematic channel in the *LogMessageCommunicationChannel* element, the details of which are shown below as a sample. The same applies when executing a *Change List* operation or any other Directory API service.

LogMessageCommunicationChannel	Element	LogMessageCommunicationChannel	0..unbounded
CommunicationChannelID	Element	CommunicationChannelID	0..1
PartyID	Element	CommunicationPartyID	0..1
ComponentID	Element	CommunicationComponentComponentID	1
ChannelID	Element	CommunicationChannelChannelID	1
LogMessageItem	Element	LogMessageItem	1
SeverityCode	Element	p1:LogItemSeverityCode	0..1
ClassificationCode	Element	LogMessageClassificationCode	0..1
Message	Element	p1:Text	0..1
languageCode	Attribute	p1:LanguageCode	optional

The UI client application should check the *LogMessageCommunicationChannel* and inform the users both when service execution is successful and when any errors are encountered.

For additional details on the *LogMessageCommunicationChannel* element as well as severity and classification codes, check the SAP Help Portal: Directory API Error Handling.

4.4 Stopping and Starting Channels

As a precaution, prior to making the switchover from one adapter engine to another, it is important that there are no longer any messages which are not in a final status (DLVD, FAIL) for communication channels targeted for switching. Otherwise, messages may become stuck in the queue after moving the channel. To reprocess these messages, the channel has to be transferred back to the original engine. This concern is more relevant for "classic" scenarios mediated through the integration engine. However, in all scenarios, it is recommended to stop targeted sender channels prior to the switch. Sender communication channels will automatically start after Adapter Engine designation is switched and activated. For a detailed description on the procedure for switching sender and receiver channels for classic scenarios, please refer to section 5.2.2 "Move Communication Channel" of the guide "How-to Handle Backlog Situations (XI 3.0/NW 7.0)".

4.5 Sending Applications

Keep in mind that applications *sending* messages to the adapter engine must change the address to the newly targeted adapter engine during the switch process (e.g. URL for sender SOAP adapter). This can be done directly in the application(s) or, in cases where a web switch or reverse proxy is used (e.g. SAP Web Dispatcher) in front of PI systems for multiple applications, the address can be changed simultaneously there. For more information on how to leverage the SAP Web Dispatcher specifically for this purpose, please check part two of the guide in this series: "How To... Fast-Switch Integration Scenarios Between SAP PI Runtimes Part II: Web Dispatcher".

4.6 Note on File Adapter (NFS)

When using Transport Protocol "File System (NFS)" for File adapters, it is recommended to set the *Source Directory* and *Target Directory* values in the Sender and Receiver channel configuration, respectively, to an **shared** location (as opposed to a **relative** location to the working directory of the configured adapter engine or an **absolute** location on the server). This will ensure that there is no dependency to the configured adapter engine when a switch takes place. If relative or absolute locations are used, then files will be processed from and to the file system associated with the local adapter engine.

For example, if based on Windows, instead of a relative location such as "/data", a shared location can be used such as "//<server>/<share>" where "//<server>/data" would be the available share location.

5. Summary

By leveraging the services of the Directory API, in a short time span, customers can have a means to fast switch the Advanced Adapter Engine configuration from one engine to another. Only 2 services of the Directory API are required to achieve this – *Communication Channel* and *Change List*. This document serves as a guide with a detailed description of the overall process for customers who desire such fast switching functionality such that they can adequately prepare and enable this functionality in their SAP NetWeaver PI landscape.

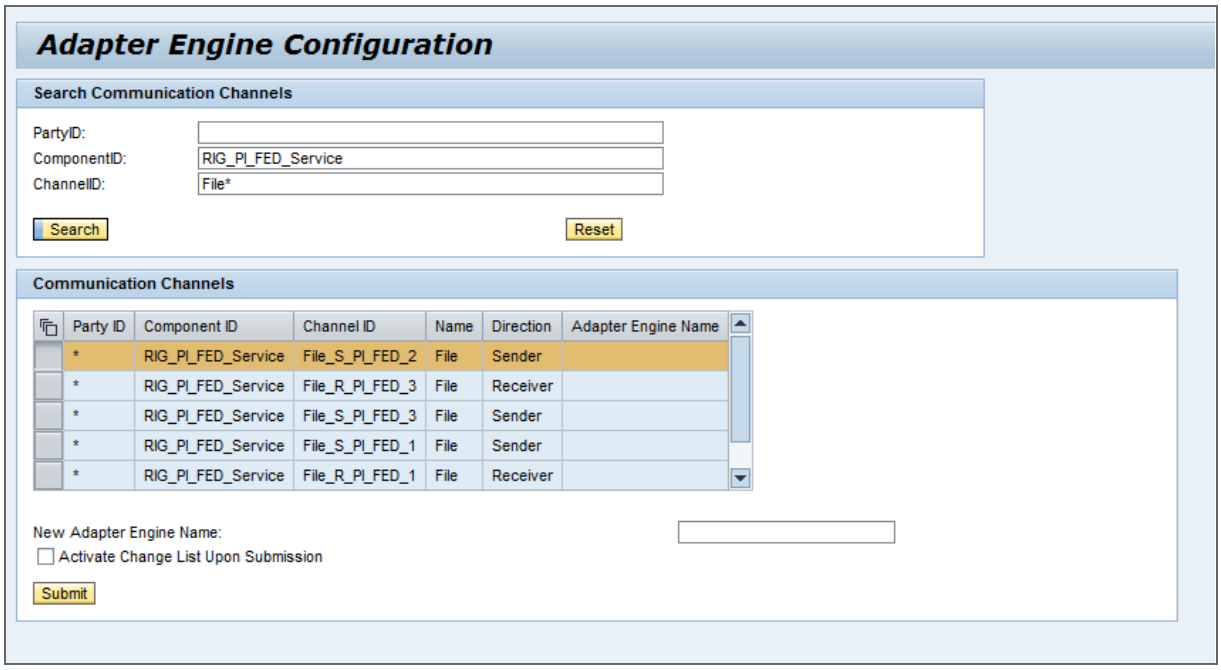
Though the version of the Directory API may be different depending upon the SAP NetWeaver PI release, the fundamental use and processing characteristics of the two main services are applicable across versions and releases.

On a final note, this guide is mainly applicable to one PI system (one SID) in the landscape. However, without too much effort, customers should be able to extend this capability to any PI system in their landscape by incorporating appropriate UI elements (e.g. drop down listing all available PI systems) that correspond to the appropriate service endpoints in the underlying service execution calls.

6. Appendix

Appendix A – Sample UI Illustration

Sample illustration of what a potential UI could look like.



www.sdn.sap.com/irj/sdn/howtoguides