

K2 connect

Course Handbook

Version: 2.2

Date: September 2016

"Terms Of Use" on page 104

Table of Contents

Table of Contents	2
Introduction to K2 connect	4
K2 connect for SAP	5
K2 SmartObjects (refresher)	6
K2 connect, SAP and SmartObjects	7
K2 SmartObjects - Architecture (refresher)	8
K2 connect Conceptual Architecture	11
K2 connect components	13
K2 connect Administration console	15
K2 connect Client console	17
The Service Object Designer	18
The Test Cockpit	20
Creating SmartObjects for K2 connect Service Objects	22
REVIEW and Q&A	23
K2 connect Configuration and Administration	24
K2 connect and SAP: Conceptual Architecture	25
K2 connect Components and Machine Boundaries	27
K2 connect: Installation	29
Destinations and Service Objects	31
Configuring Destinations	34
Multiple Destinations	38
Isolated SAP Systems and Destinations	39
Shared SAP Systems and Destinations	39
Destination Naming	40
Service Object Explorer and Test Cockpit	42
Design-time Authentication	45
Runtime Authentication	47
K2 SSO and SAP Authentication	51
Distributed Install - NLB	56
Logging and Troubleshooting	61

Review and Q&A	64
Developing with K2 connect	66
Service Objects and SmartObjects	67
K2 connect development tools	69
The Service Object Designer	71
Test Cockpit	74
High-level steps to expose a SAP BAPI as a SmartObject	77
Service Methods, Parameters and Structures	81
Custom Service Methods	85
Using XML properties instead of Structures	88
What happens when you publish a Service Object?	91
Writing data to SAP	93
Custom Transformation Mapping Code	95
Deploying to other servers/environments	97
Review and Q&A	102
Terms Of Use	104
ACCEPTANCE OF TERMS	104
COPYRIGHT	104
THIRD-PARTY INTEGRATION	104
NOTICE REGARDING CONTENT OF THIS DOCUMENTATION	104

Introduction to K2 connect



This learning module introduces K2 connect and describes the basic architecture used to expose SAP BAPIs as K2 SmartObjects. This module is intended for all roles that will plan, install, develop with, or administer K2 connect. Typical roles include K2 administrators and infrastructure administrators, Solution architects, SAP Developers, and Administrators and K2 Developers

At the end of this module, participants should have a basic understanding of K2 connect and how it can expose SAP BAPIs as K2 SmartObjects, This is just an introductory module: participants will need to complete one or more of the subsequent K2 connect learning modules before they will be able to use K2 connect effectively in a real-world installation.

Tip

A Recorded video of this module is available at <https://youtu.be/DsnNY6AALqc>

K2 connect for SAP

K2 connect for SAP

- K2 connect is an add-on for K2 blackpearl
- Allows K2 designers and developers to visually create reusable business entities from SAP BAPIs without writing any code
- Consume these reusable business entities from any technology that supports K2 SmartObjects (Workflows, Forms, Reports, etc.)
- K2 connect acts as an intermediary between K2 blackpearl SmartObjects and SAP BAPIs
- In K2 4.6.5 and later, the K2 Endpoints Service broker allows you to interact with SAP Web Services



K2 connect is an add-on component for K2 blackpearl. The current implementation of K2 connect allows K2 designers and developers to visually create reusable business entities (SmartObjects) from SAP BAPIs without writing any code.

These SmartObjects can then be consumed in any technology capable of interacting with K2, including K2 workflows, K2 smartforms, report design tools, .NET code and more. Essentially, K2 connect makes it possible for K2 application designers to interact with SAP BAPIs through SmartObjects without knowing anything about SAP or needing to interact with any SAP interfaces or BAPIs.

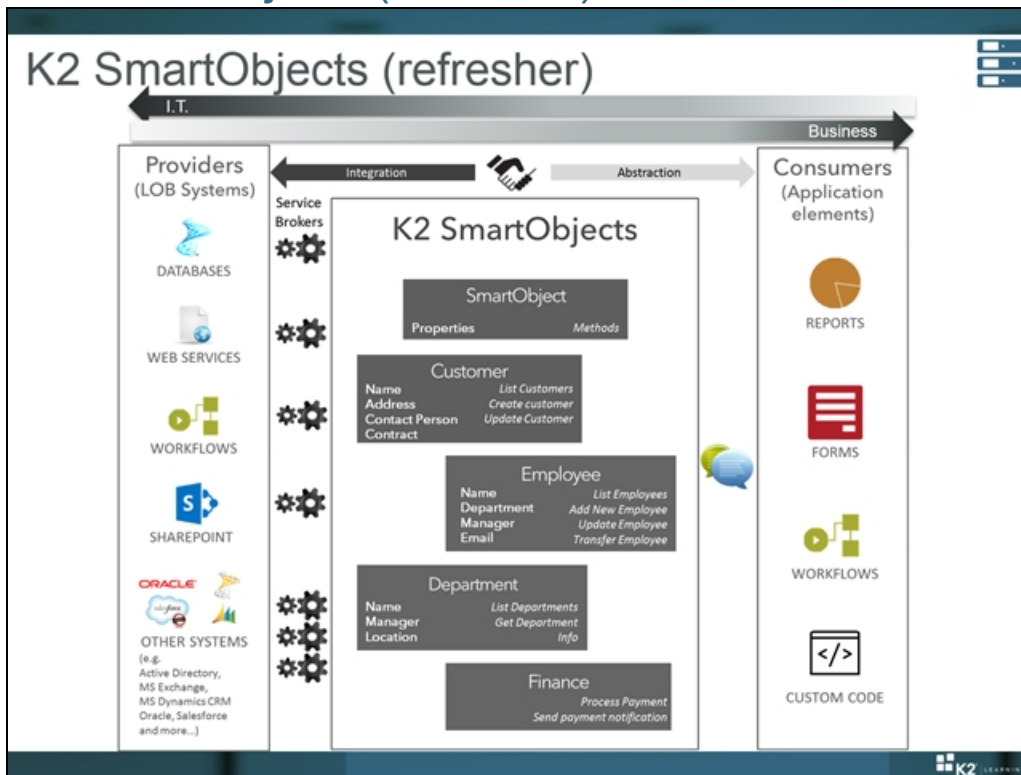
Developers who design K2 connect service objects for SAP BAPIs should have an understanding of the BAPIs that will be used, and troubleshooting may require some SAP knowledge, but beyond that, K2 connect does not require any advanced SAP knowledge or expertise.

K2 connect's purpose is to act as an intermediary between K2 blackpearl and SAP systems. It merely performs processing and translation tasks.

Note

Note that K2 connect only allows you to interact with SAP BAPIs. If you are using K2 4.6.5 or later, the new Endpoints Service Broker will allow you to interact with SAP Web Services (and many more services, besides).

K2 SmartObjects (refresher)



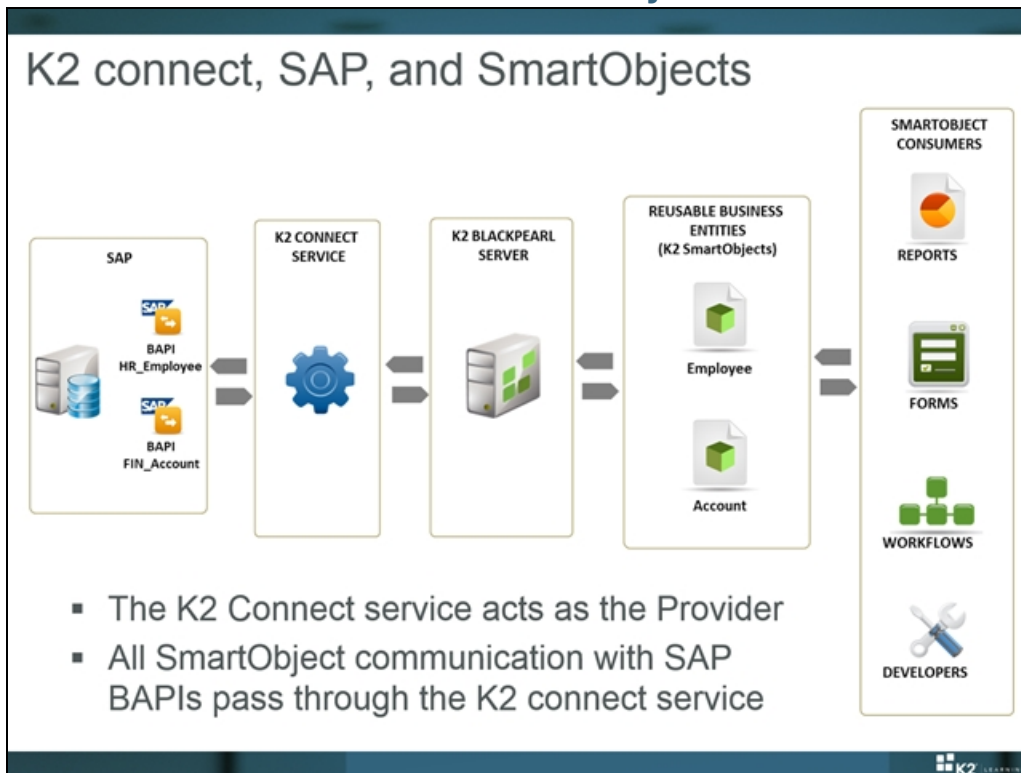
K2 SmartObjects are an integral part of K2 connect, since they are the objects that are eventually surfaced to SmartObject consumers like workflows or forms. If necessary, take a moment to refresh your understanding of SmartObjects, since the remainder of this module assumes that you are familiar with K2 SmartObjects.

You can think of SmartObjects as a repository of business objects, which expose various back-end systems as logical entities. As far as the K2 application designer is concerned, they have access to a repository of business objects that they can use in various technologies. The designers do not need to know where the data comes from or how to integrate with the various back-end systems.

You may recall that SmartObjects have consistent property types and method types, regardless of the underlying technology or data provider. This consistent behavior allows consuming applications to interact with business objects in the same way, regardless of the underlying technology.

The diagram in the slide for this topic illustrates this concept: SmartObjects are effectively an abstraction layer, which exposes various back-end systems and data providers as logical business objects to consuming applications. (The diagram above just lists just a few examples of providers and consumers. You could expose many other providers as SmartObjects, and consume SmartObjects from many other applications. For now, it's only important to understand the abstraction and re-use capabilities of SmartObjects).

K2 connect, SAP and SmartObjects

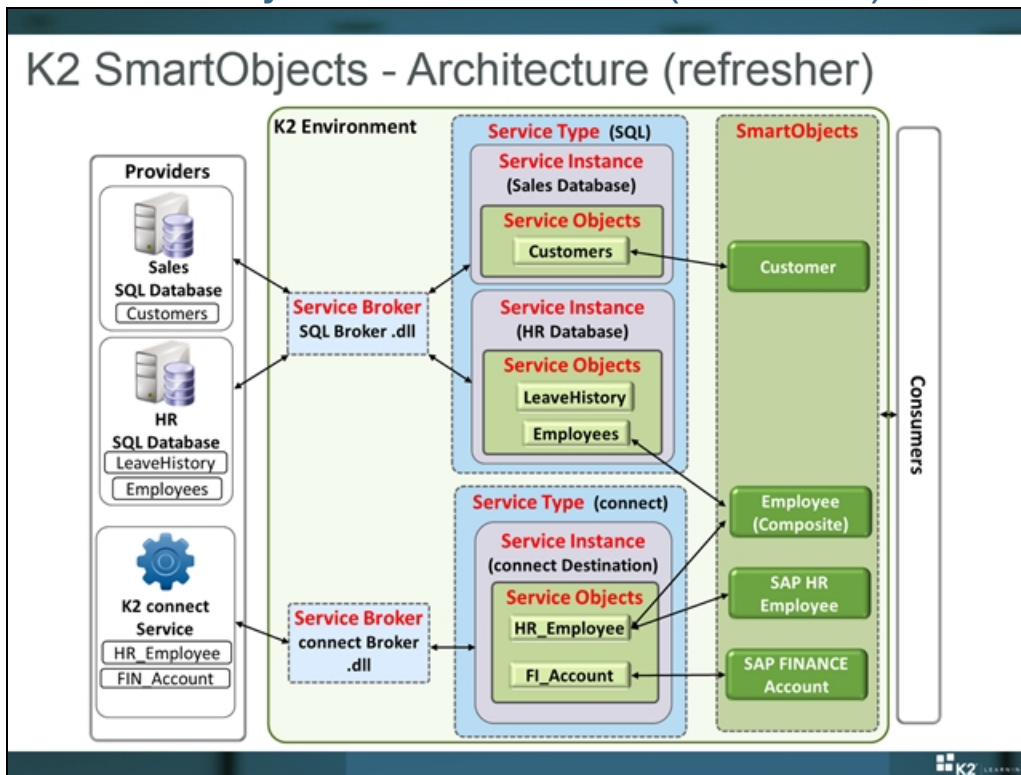


K2 connect is a separate service which acts like a Provider for SmartObject data. Consider the diagram in the slide for this topic. You will notice that we represent the familiar **SmartObjects** and **Consumers** relationship on the right side of the diagram. Remember that SmartObjects are hosted and executed by the K2 blackpearl server.

On the left side of the diagram, we have a hypothetical SAP system with two BAPIs (one to return Employee information from the HR module and another that returns Account information from the Finance module). We need the intermediary K2 connect service, which is essentially a Provider for the blackpearl server, to expose these BAPIs to K2 blackpearl. The K2 connect service performs all the necessary processing and translation that is required to interact with the SAP system. (The processing is based on definitions that were created by a developer with K2 connect design tools. We will look at those tools shortly). Eventually, these BAPIs are exposed as two SmartObjects (Employee and Account) which can be consumed in various technologies.

Why is this diagram important? Because K2 connect behaves a little differently than a standard Service Broker. Instead of K2 blackpearl connecting directly to SAP, there is an intermediary service (K2 connect) which performs the processing. We'll look at this in more detail in the following topics.

K2 SmartObjects - Architecture (refresher)



To understand the behavior of K2 connect, take a moment to refresh your understanding of the internal architecture of SmartObjects, and specifically the various terms like "Service Objects", "Service Instance", and "Service Broker".

Refer to the diagram in the slide for this topic (a larger version follows below this text). In the diagram, the left-hand side (**Providers**) represents the underlying data sources that need to be consumed by the **Consumers** on the right side. The green area between the Providers and Consumers is essentially SmartObjects: think of it as a translator, adapter, or middle-layer between the Providers and the Consumers. Everything in this green area between the provider and consumers happens on the K2 Server.

Inside the **K2 Environment** box we have **Service Brokers**, **Service Types**, **Service Instances**, **Service Objects** and **SmartObjects**. These are the components that work together to make up the SmartObject component of K2.

Service Broker

A Service Broker is a .dll file which contains all the code necessary to interact with a specific provider. Typically, this assembly would reference an API or Service exposed by the provider, and it would contain code that discovers and/or describes that provider's objects to K2 in a consistent way. Service Brokers are normally written to expose a particular technology. In our example diagram, we have two different Service Brokers: the SQL Broker.dll contains all the logic necessary to interact with Microsoft SQL databases, while the *connect Broker.dll* knows how to interact with a K2 connect service. Think of Service Brokers as system-specific connectors.

Service Type

The purpose of a *Service Type* is to register a Service Broker .dll file in a K2 environment so that it can be used to register Service Instances.

Service Instance

Once a Service Broker is registered as a Service Type, K2 Administrators can create one or more Service Instances for that Service Type. A Service Instance is essentially a configured instance of a broker, and contains various configuration values that are required by the Broker. The configuration values could include values that the Broker needs to connect to the provider (for example, a server name) and perhaps additional configuration values that describe how the Broker should behave. Another important setting for a Service Type is the Authentication Mode setting: this determines the user credentials that are passed to the underlying Service Broker at runtime.

In our sample diagram, there is a Service Instance for the connect broker which points to a specific connect destination (we will talk about Destinations a little later), while the SQL service has two instances: one which connects to the Sales database and another connecting to the HR database.

Service Object

Service Objects are essentially "translated" representations of the entities in the underlying Provider. Under the covers, Service Objects are just XML representations of the entities in the provider and contain no processing logic.

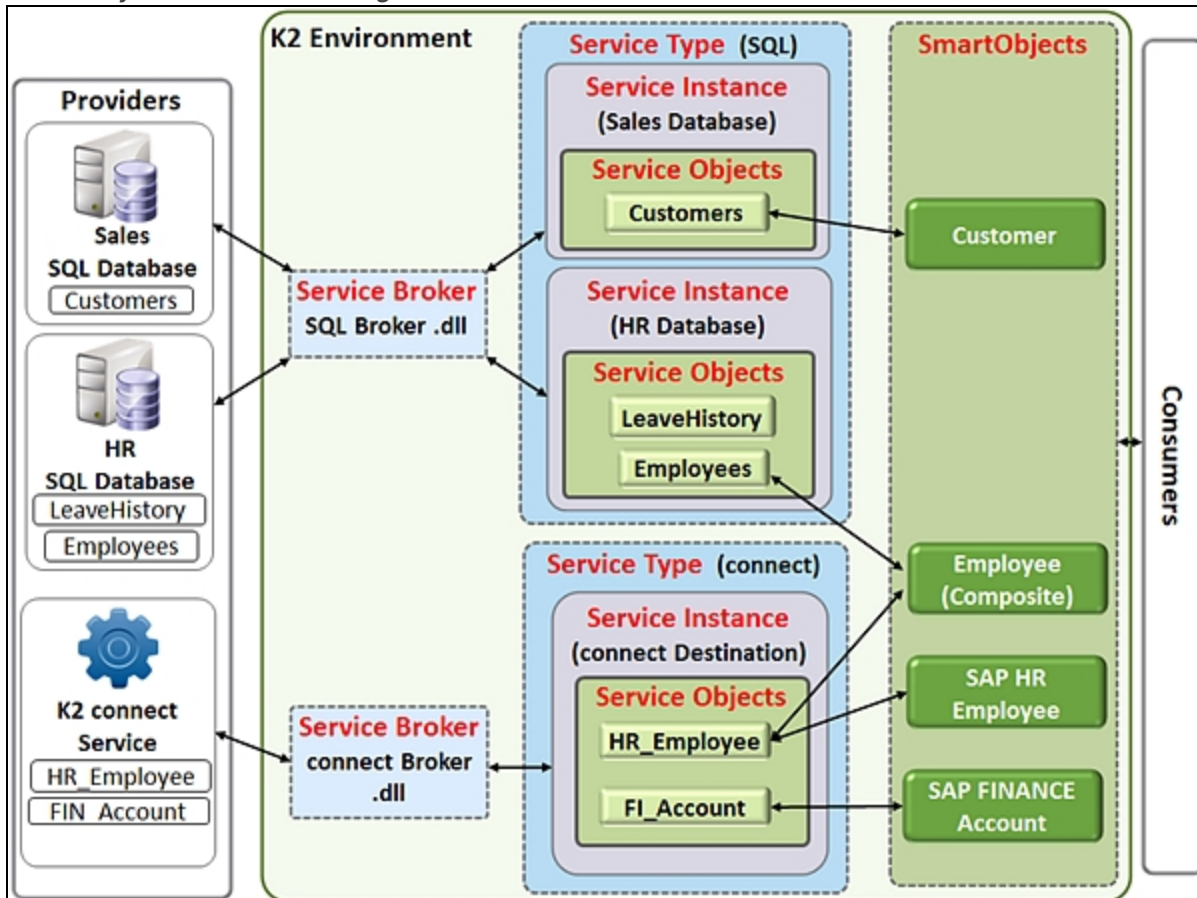
Service Objects expose the properties and methods for objects in a Provider as consistent SmartObject Method Types and Property Types; it is the responsibility of the Service Broker to perform the necessary mapping between the Provider's types and the types used by SmartObjects.

In our sample diagram, the connect Service instance has two Service Objects (*HR_Employee* and *FI_Account*) which point to the objects in the K2 connect service. The SQL Service Instances have Service Objects for each of the tables in the respective databases.

SmartObjects

SmartObjects are the reusable business entities that consumers will interact with. Remember that you can create composite SmartObjects which combine data from different systems (in the diagram, the *Employee* SmartObject combines data from SAP and SQL) and you can expose the same Service Object as different SmartObjects (in the diagram, both the *Employee* and *SAP HR Employee* SmartObjects use the same *HR_Employee* Service Object).

SmartObject Architecture Diagram



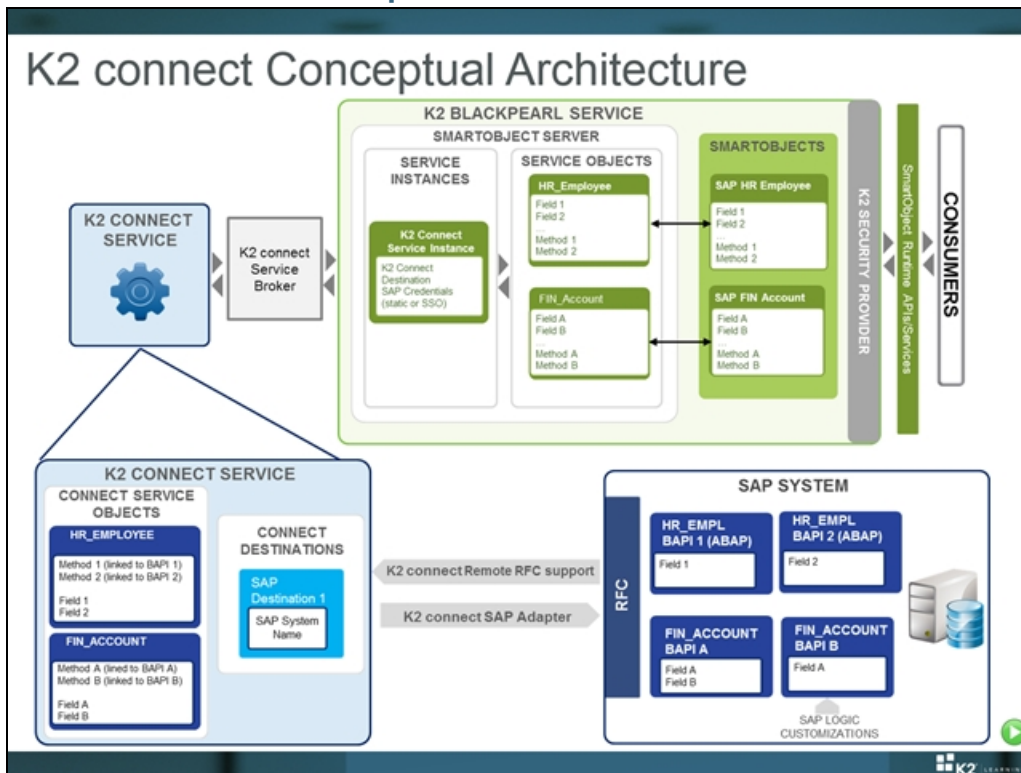
Why is all this important for K2 connect?

From a Service Instance and Service Object perspective, K2 connect provides the definitions to K2 blackpearl. These definitions are created by a developer using K2 connect development tools, and then are published to the K2 connect server.

This implies that the K2 connect Service Broker .dll in the blackpearl service does not actually connect directly to a SAP environment to discover the available BAPIs: instead, it connects to the K2 connect service which, in turn, uses the definitions published by a developer to represent SAP BAPIs. Applying this principle to the diagram above, it's actually the K2 connect Service which is a Provider.

We will dive a little deeper into the K2 connect architecture in the next topic. For now, just understand the fundamental purpose of the various components in the SmartObject architecture.

K2 connect Conceptual Architecture



Now that you understand that the K2 connect service is a Provider for SmartObjects, let's look at the architecture of the connect service to understand how SAP BAPIs are surfaced as SmartObject Service Objects.

Consider the diagram in the slide for this topic (a larger version of the diagram is provided below). The top half of the diagram represents the information we covered in the previous topic: how Service Instances, Service Objects and SmartObjects work together, and that K2 connect is the Provider for SmartObjects that ultimately interact with a SAP system.

In the top half of the diagram, notice that we have two SmartObjects: **SAP HR Employee** and **SAP FIN Account**. Each SmartObject has two methods and some fields/properties. The methods are linked to separate SAP BAPIs, while the fields represent the data returned by the BAPIs.

Next, let's look at the K2 connect service in more detail. In our example, the **K2 connect service** contains two connect Service Objects (**HR_EMPLOYEE** and **FIN_ACCOUNT**). Each of these Service Objects contains two methods which are linked to specific SAP BAPIs. The service objects were created by a developer using the connect development tools.

Note

Connect *Service Objects* are different from *SmartObject Service Objects*. It is just a coincidence that they use similar names, but the object definitions are actually separate and stored separately. It's easy to get confused between the two, so for clarity we will always refer to *connect Service Objects* when talking about the object definitions stored in the connect Service.

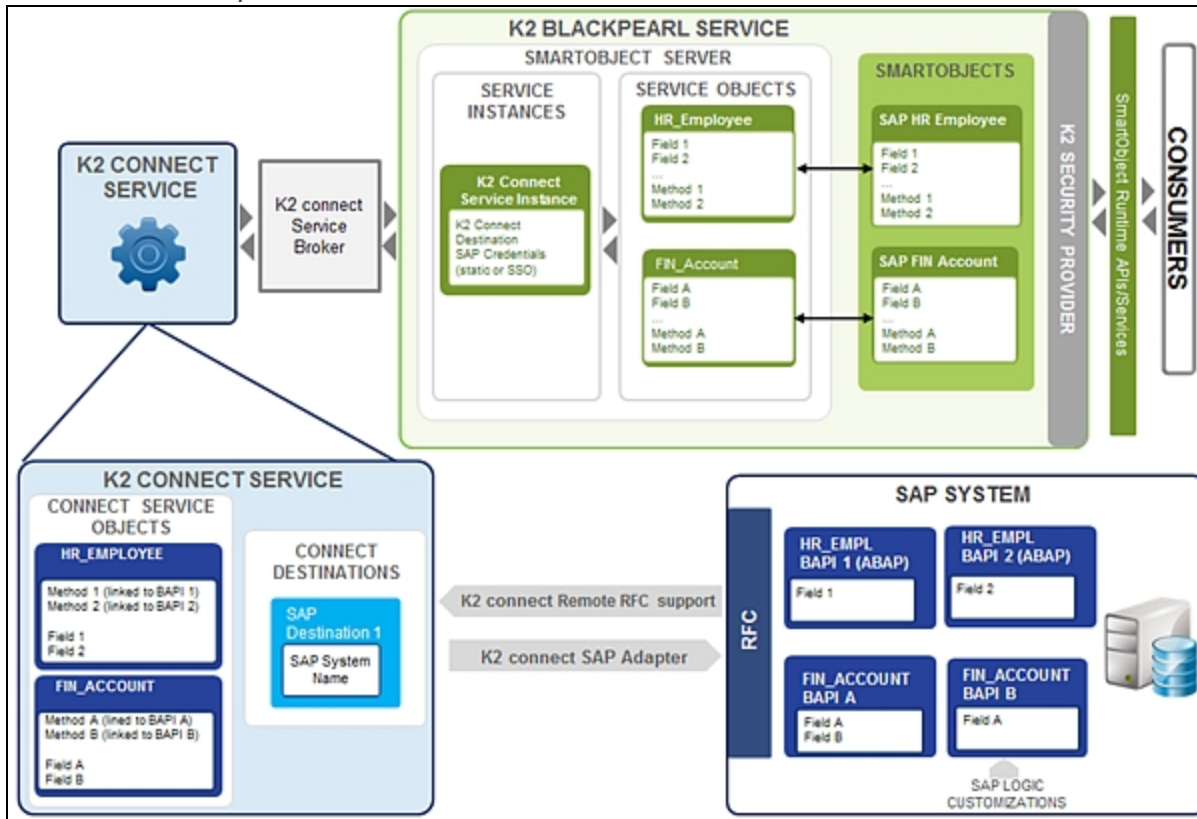
The next part in the connect Service is the **connect Destination**. This is similar to the concept of a Service Instance, but it's not the same thing. A connect Destination is a configuration which tells the connect service how to connect to a particular SAP system. Usually the K2 administrator sets up this Destination, and developers will use this Destination to connect to a particular SAP system. In our sample diagram, we have a single destination called **SAP Destination 1**, and this destination is configured with the connection details required to reach the target SAP System.

(It is quite possible that a K2 connect service could contain multiple destinations, each pointing to a different SAP system.)

The connect service interacts with the SAP system using a third-party adapter called **ERP connect**, which executes SAP BAPIs using Remote Function Calls (RFC).

In our example diagram, the SAP system contains four different BAPIs. **BAPI 1** and **BAPI 2** are methods in the **HR_EMPL** business object, while **BAPI A** and **BAPI B** are methods in the **FIN_ACCOUNT** business object. (Note that BAPIs are not objects: they are just functions). These BAPIs have been mapped by a developer to methods in the respective **HR_EMPLOYEE** and **FIN_ACCOUNT** connect Service Objects. Don't be too concerned about the details just yet: the next learning module will go into the configuration and set-up in much more detail. For now, it's sufficient to understand the basic conceptual architecture of K2 connect and how it represents SAP BAPIs.


K2 connect Conceptual Architecture



K2 connect components

K2 connect Components

- K2 connect service
 - Windows service
 - Installed on each physical K2 Server
 - Configured with the administration console
- K2 connect database
 - SQL database
 - Stores connect configuration and service object definitions
- K2 connect administration console
 - Primary connect administration tool
- K2 connect client console
 - Deploy connect Service Objects without Visual Studio
- Service Object Designer (Visual Studio)
 - Developer tool used to discover and map BAPIs
- Test Cockpit (Visual Studio)
 - Developer tool used to test BAPIs



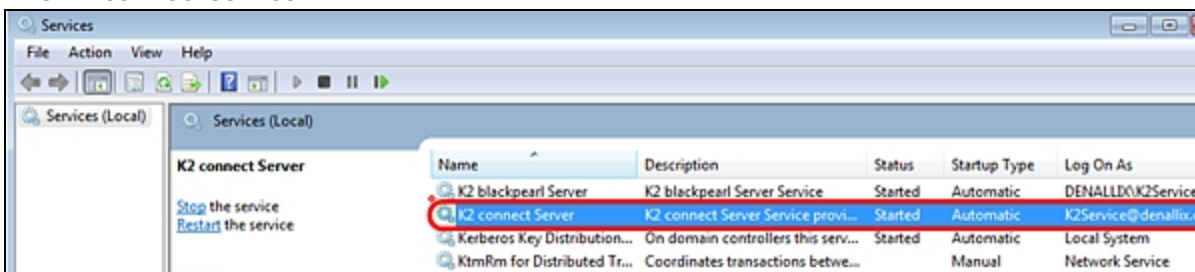
K2 connect has several components, some of which are installed on the K2 server and some of which are installed on developer workstations. We will briefly list the components, and the next few topics will explain some of these components in more detail.

K2 connect service

The K2 connect server runs as a separate Windows service and is installed on the same physical server as the K2 service.

To simplify things somewhat, this service performs the conversion between K2 Service Objects and the ERPConnect SAP adapter. It also stores and serves the connect Service Object definitions. This service may or may not use the same service account as the K2 blackpearl service.

The K2 connect service



K2 connect database

K2 connect has its own dedicated SQL database, separate from the K2 blackpearl database(s). This database stores the connect configuration information, the connect service object definitions and other internal settings. This database does not have to exist on the same SQL instance as the K2 databases, but it simplifies administration somewhat if the databases exist on the same SQL server instance. Depending on the number of connect Service Objects and how heavy the runtime load is, this database will probably take up few resources on the SQL server and it should be safe to install it on the same instance as the K2 databases.

You should never need to make any manual changes in this database, and manual updates to this database are not supported nor recommended by K2 unless you are specifically directed to do so by a K2 representative.

Connect Administration console

Administrators (and sometimes developers) use the connect Administration console to configure the K2 connect service and to set up connections (Destinations) to SAP environments. It can also be used to register service object definitions (.svd's) manually. This component is normally installed on the connect server, administrator workstations, and on developer workstations.

Connect client console

The connect client console is not used that often, but it allows the user to remotely refresh and add or remove K2 connect client interfaces without using Visual Studio. In most cases, the connect Administration console or Visual Studio is used to publish Service objects, but this tool is available for environments with strict change control policies.

Service Object Designer

The Service Object Designer is a plug-in for Visual Studio that allows developers to explore the BAPIs available in SAP systems and expose them as connect Destinations. This component is installed on developer workstations, and Visual Studio is a prerequisite for this component.

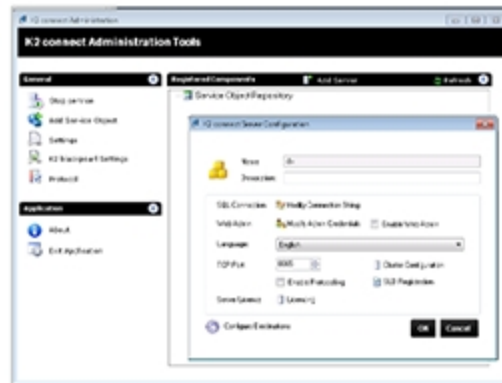
Test Cockpit

The Test Cockpit is a tool which allows developers to test and explore SAP BAPIs without committing any data. It is installed on developer workstations along with the Service Object designer, and like the Service Object Designer, runs inside Visual Studio.

K2 connect Administration console

K2 connect Administration Console

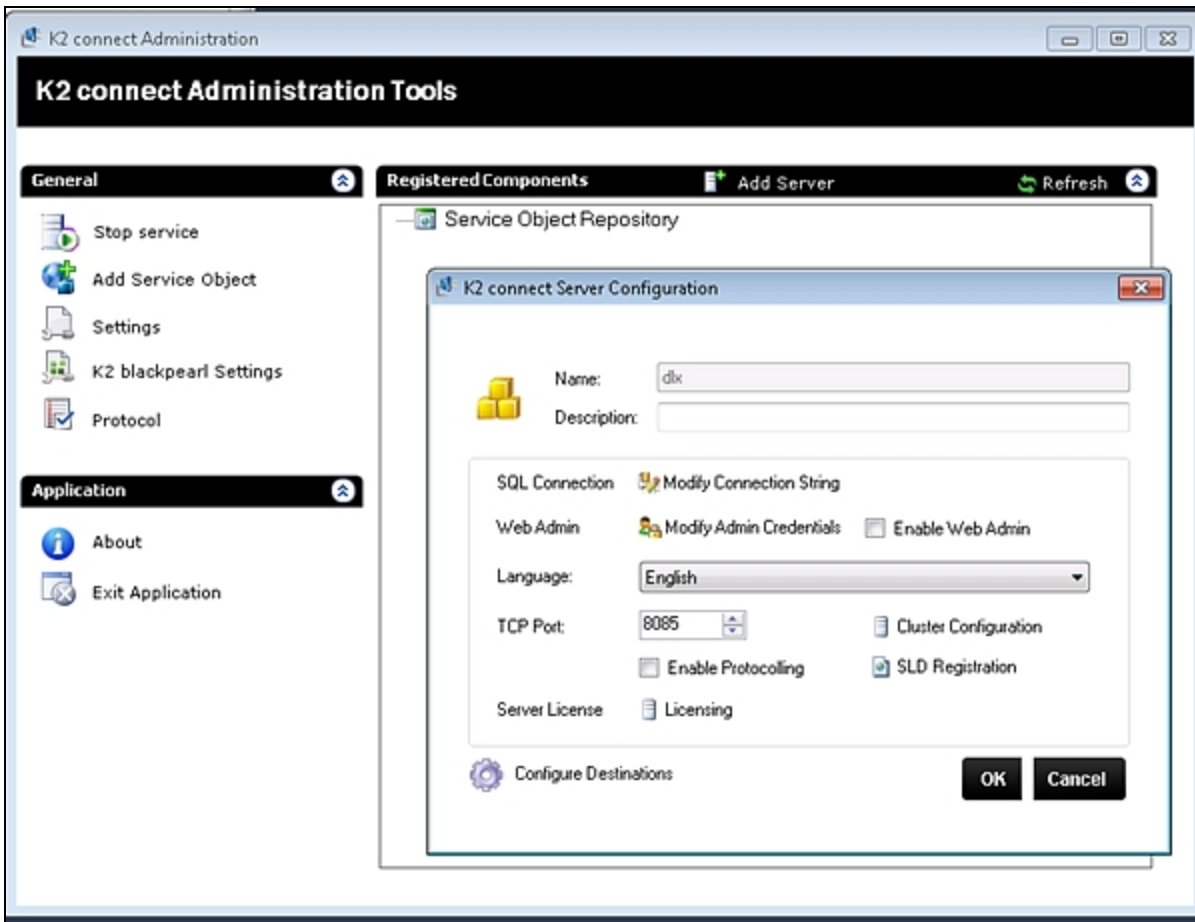
- Stand-alone utility
- Configure connect Service
- Configure licensing
- Configure destinations
- Define static credentials for SAP connections
- Cluster configuration
- Manual import of Service Object definitions (.svd's) or client interfaces (.dll's)
- Advanced configuration



The connect administration console is a vital tool for administrators and developers. It allows these users to configure the connect service, view and update connect licensing, configure SAP systems as Destinations and, if necessary, define static credentials for connecting to SAP environments.

This tool can also be used to import service object definitions (as .svd's or proprietary client interface .dlls) manually. Administrators can also use this tool to set up configurations such as cluster configuration and change the more advanced configuration settings.


We will cover this tool in more detail in the module 300.PEK K2 connect configuration and administration.



K2 connect Client console

K2 connect Client Console

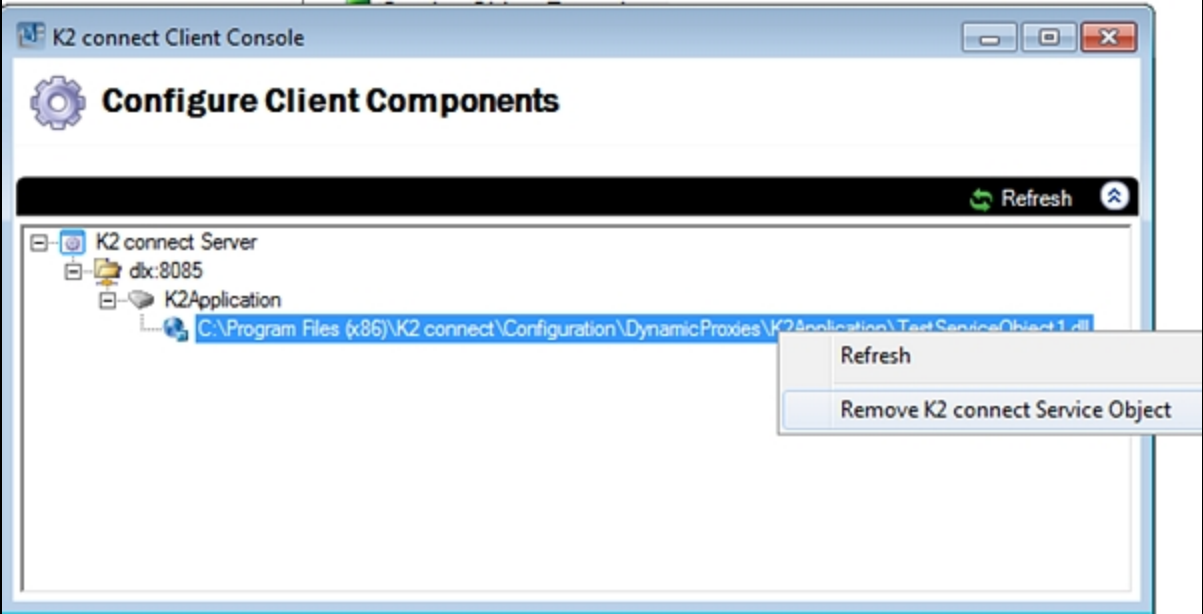
- View and manage K2 connect client interfaces (.dll's)
- Publish connect Service Objects without Visual Studio



The screenshot shows the 'K2 connect Client Console' window. The title bar reads 'K2 connect Client Console'. The main area is titled 'Configure Client Components'. Below this, there is a 'Refresh' button with a circular arrow icon. A tree view shows the following structure: 'K2 connect Server' (expanded) -> 'dx:8085' (expanded) -> 'K2Application' (selected). A context menu is open over 'K2Application', showing two options: 'Refresh' and 'Remove K2 connect Service Object'. The K2 logo is visible in the bottom right corner of the window.

The K2 connect client console's primary purpose is to allow users to remotely refresh, update, or delete service object definitions on a K2 connect server. (Note: The client console only supports using client interface .dlls, not .svd files.) This tool is normally used when the organization has strict change control policies and does not want to install the connect administration console or Visual Studio tooling on the machine of the user who is responsible for publishing the connect Service Objects.

The K2 connect Client console

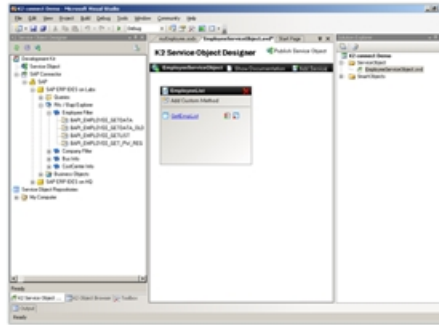


This is a detailed screenshot of the 'K2 connect Client Console' window. The title bar shows 'K2 connect Client Console' with standard window controls. The main content area is titled 'Configure Client Components' with a gear icon. Below the title, there is a 'Refresh' button with a circular arrow icon. A tree view shows the following structure: 'K2 connect Server' (expanded) -> 'dx:8085' (expanded) -> 'K2Application' (selected). A context menu is open over 'K2Application', showing two options: 'Refresh' and 'Remove K2 connect Service Object'. The path for the selected item is shown as 'C:\Program Files (x86)\K2 connect\Configuration\DynamicProxies\K2Application\Test Service Object 1.dll'.

The Service Object Designer

The Service Object Designer

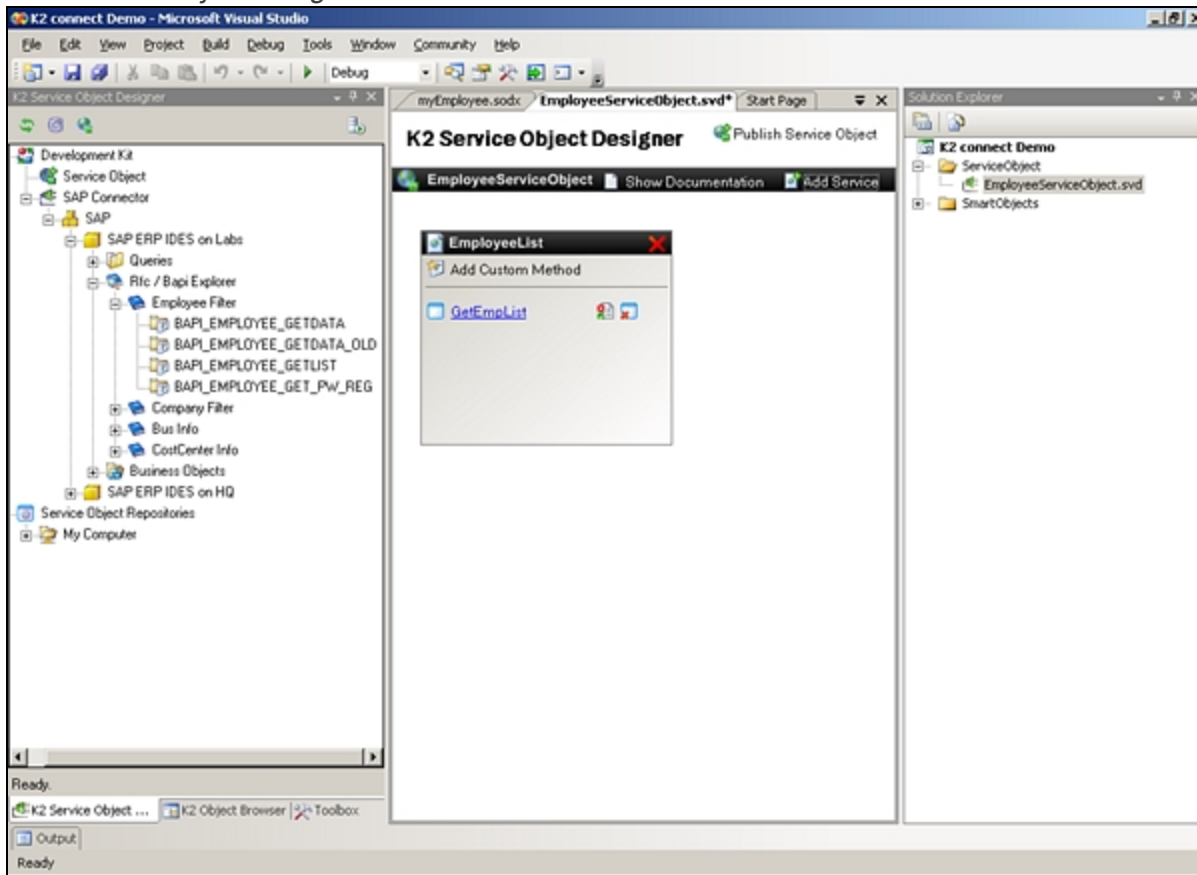
- Add-in to Microsoft Visual Studio
- Visually explore the BAPIs exposed in a SAP environment
- Drag-and-drop configuration of Services
- Visually map input and output properties on the Service Methods



The Service Object designer is an add-in for Microsoft Visual Studio. Developers use it to design connect Service Objects using a visual, drag-and-drop interface.

The designer exposes a tree-like explorer component which is used to explore the available BAPIs in a particular SAP system. The target SAP systems are exposed as connect Destinations. Developers can use the explorer to register additional Destinations if they have not already been set up by the administrator.

The Service Object Designer



The Test Cockpit

The Test Cockpit

- Utility launched from the Service Object Designer to explore BAPIs
- Explore the input and output parameters or structure of functions without modifying data



The Test Cockpit is a utility which is installed along with the Service Object designer in Visual Studio. Developers use it to explore and execute specific BAPIs to discover the input and output parameters of the BAPIs without modifying any data in the underlying SAP system.

The Test Cockpit

K2 [connect] Test Cockpit

Options

Destination: SAP Connector | SAP | SAP ERP IDES on Labs

User Name: Language:

Password:

Predefined User Credentials

Function Name: BAPI_EMPLOYEE_GETDATA

Information

Elapsed time: 1121 ms

Ready

Load Interface Execute

Input

DATE

EMPLOYEE_I...

EXTENSION

FSTNAME_M

FST_NAME_K

FST_NAME_R

JOBTXT

JOBTXT_IG

Rollback Commit Work Enable ABAP Debugger

Output

RETURN [Show Details...](#)

ARCHIVELIN... [Show Details...](#)

COMMUNICAT... [Show Details...](#)

INTERNAL_C... [Show Details...](#)

ORG_ASSIGN... [Show Details...](#)

PERSONAL_D... [Show Details...](#)

Table: PERSONAL_DATA

LAST_NAME	LAST_NAME2	FIRSTNAME	TITLE	TITLE_2	ARI_TITLE	NAMEAFFIX	NAMEPREFIX	KNOWN...
Paulsen		Olaf						
Kerze		Olaf						
Hansen		Olaf						

Creating SmartObjects for K2 connect Service Objects

Creating SmartObjects for K2 connect Service Objects

1. Developers publish connect Service Objects to the K2 connect service using the Service Object Designer
or
Administrators import .svd files with the K2 administration tools
2. K2 connect Service Objects are generated
3. Manual or automatic update of the connect Service Instance on the K2 server, which creates/updates the Service Instance's Service Objects
4. Designers can use K2 tools to design/update simple or composite SmartObjects using the connect Service Instance's Service Objects
 - K2 Studio
 - K2 for Visual Studio
 - K2 Designer
 - SmartObject Service Tester utility



If you think back to the SmartObjects diagram from the start of this learning module, you will recall that a SmartObject is associated with one or more Service Objects. There are several ways that these SmartObjects may be created, but fundamentally a SmartObject is always associated with at least one K2 Service Object in a Service Instance. You will recall from the K2 connect architecture discussion that the K2 blackpearl server never interacts directly with the SAP system; instead, it uses the connect Service Objects which have been published to the K2 connect service.

From a high-level perspective, the publishing process works something like this:

1. Developers use the Service Object Designer in Visual Studio to define a K2 connect Service Object, which points to one or more SAP BAPIs. This results in a .svd file which developers may publish directly to a K2 connect server, or they may hand the .svd file to an administrator who will import the Service Object Definition file into a K2 connect server
2. Connect Service Objects are automatically generated in the connect service when the Service Object definition is published.
3. Depending on the interface used to publish the Service Object definition, the K2 connect service may be restarted and will update the associated connect Service Instance on the K2 blackpearl server, or these steps may be performed manually. This process will create new (or update existing) Service Instance's Service Objects for each of the connect Service Objects in the connect service.
4. Once the K2 Service Objects are updated, designers can use any of the standard K2 tools (e.g. K2 Studio, K2 for Visual Studio, K2 Designer, or the SmartObject Service Tester utility) to create simple or composite SmartObjects which interact with the Service Objects. Notice that the designers never interact directly with the SAP environment or the K2 connect service: as far as they are concerned, the Service Objects that point to K2 connect are just like any other Service Object.

Note

Although the SmartObject Service Tester utility makes it very easy and quick to generate SmartObjects, we recommend using this approach for testing purposes only. When you are creating real SmartObjects it is better to use one of the other design tools so that you have granular control over the SmartObject name and methods.

REVIEW and Q&A

Review and Q&A

- K2 SmartObjects refresher
- K2 connect and SAP
- The conceptual architecture of K2 connect, SAP BAPIs, and K2 SmartObjects
- Connect Administration tools
- Connect Developer tools and utilities
- Basics of creating SmartObjects for SAP BAPIs

Let's review the main topics covered in this learning module:

- K2 SmartObjects refresher
- K2 connect and SAP
- The conceptual architecture of K2 connect, SAP BAPIs and K2 SmartObjects
- connect Administration tools
- connect Developer tools and utilities
- Basics of creating SmartObjects for SAP BAPIs

Think about your own SAP systems and the BAPIs you may want to expose as SmartObjects. Can you identify any potential composite SmartObjects that would combine data from SAP and other systems? How would K2 designers use these SmartObjects?

If you are in a class setting, please ask the instructor to explain or elaborate on topics you don't fully understand. If you are in a self-study situation, check out the additional resources listed below. The next two modules will cover K2 connect and SAP integration in much more detail, though, so your questions may be deferred until you have completed those modules.

Additional Resources

The following table lists additional resources that supplement the information provided in this module:

Resource	Location and Notes
SAP Integration Overview	http://www.k2.com/platform/integration/sap/ <i>SAP integration overview on K2.com</i>
K2 connect Product Documentation	http://help.k2.com/en/connectforsap.aspx <i>Online version of the K2 connect product documentation</i>
K2 connect on K2 underground	http://community.k2.com/t5/forums/searchpage/tab/message?filter=labels&q=k2+connect <i>K2 connect group on the K2 Community forums</i>

K2 connect Configuration and Administration



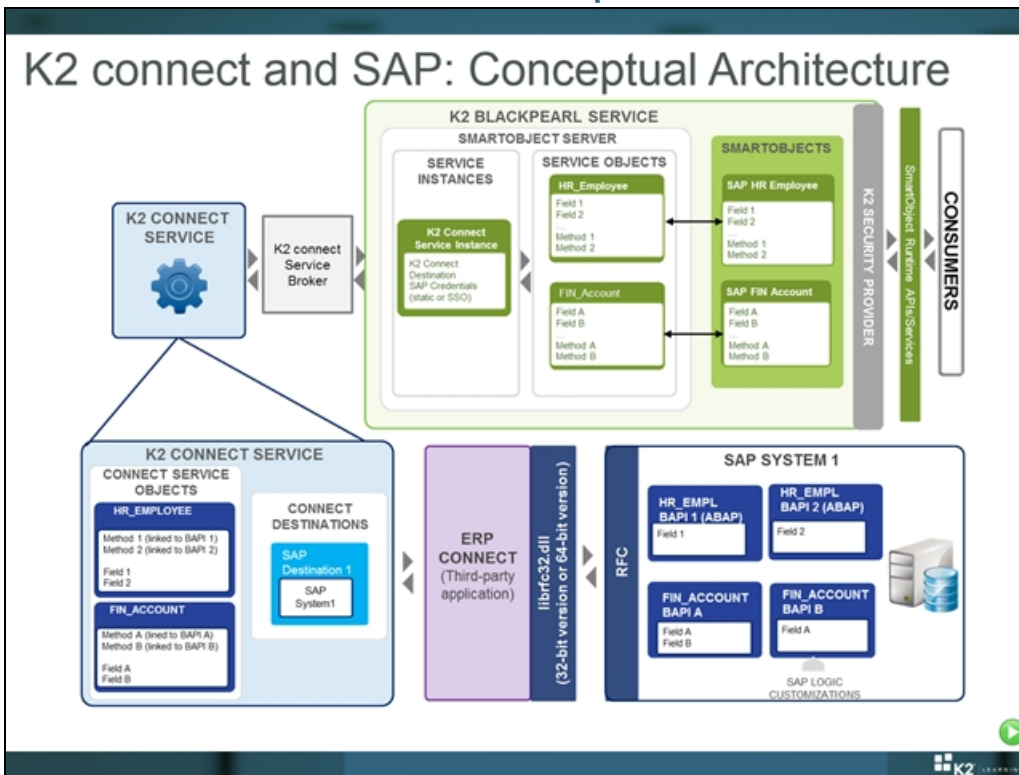
This learning module describes how K2 connect is installed and how K2 connect installations are configured, administered, and tested. Coverage includes installation and configuration of K2 connect, K2 connect destinations, and authentication options. Participants will also learn how to test the integration between K2 connect and SAP and how to troubleshoot issues with available logging tools. This module is intended for all roles that will plan, install, develop with or administer K2 connect. Typical roles include K2 administrators and infrastructure administrators, K2 Developers, Solution architects, SAP Developers and SAP Administrators.

At the end of this module, participants should have a solid understanding of how to install and configure K2 connect to expose a particular SAP system as a connect Destination. Participants should also be able to perform basic tests to verify that the integration is working as expected, and understand what tools are available for troubleshooting when it does not.

Tip

A Recorded video of this module is available at <https://youtu.be/du9nu4eFEKo>

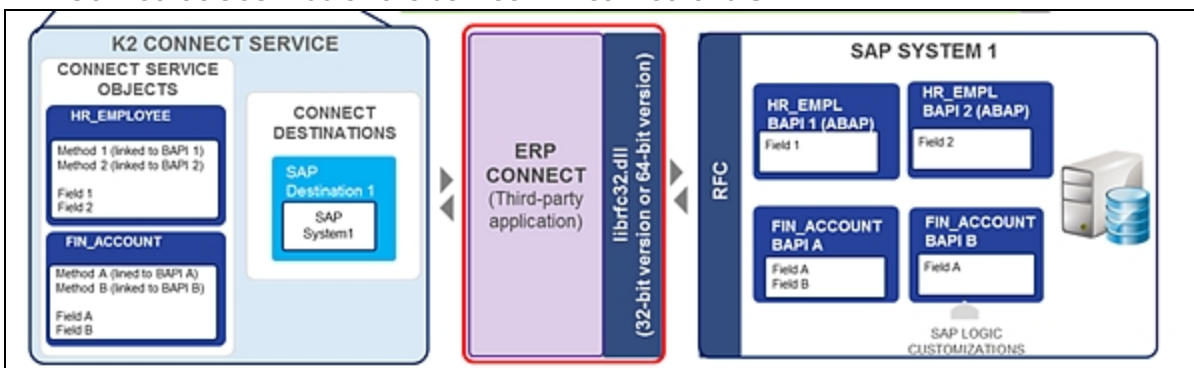
K2 connect and SAP: Conceptual Architecture



We previously discussed the basic conceptual architecture of K2 connect in the learning module **200.PVG - Introduction to K2 connect**. Let's expand on that knowledge and describe some of the internal workings of K2 connect.

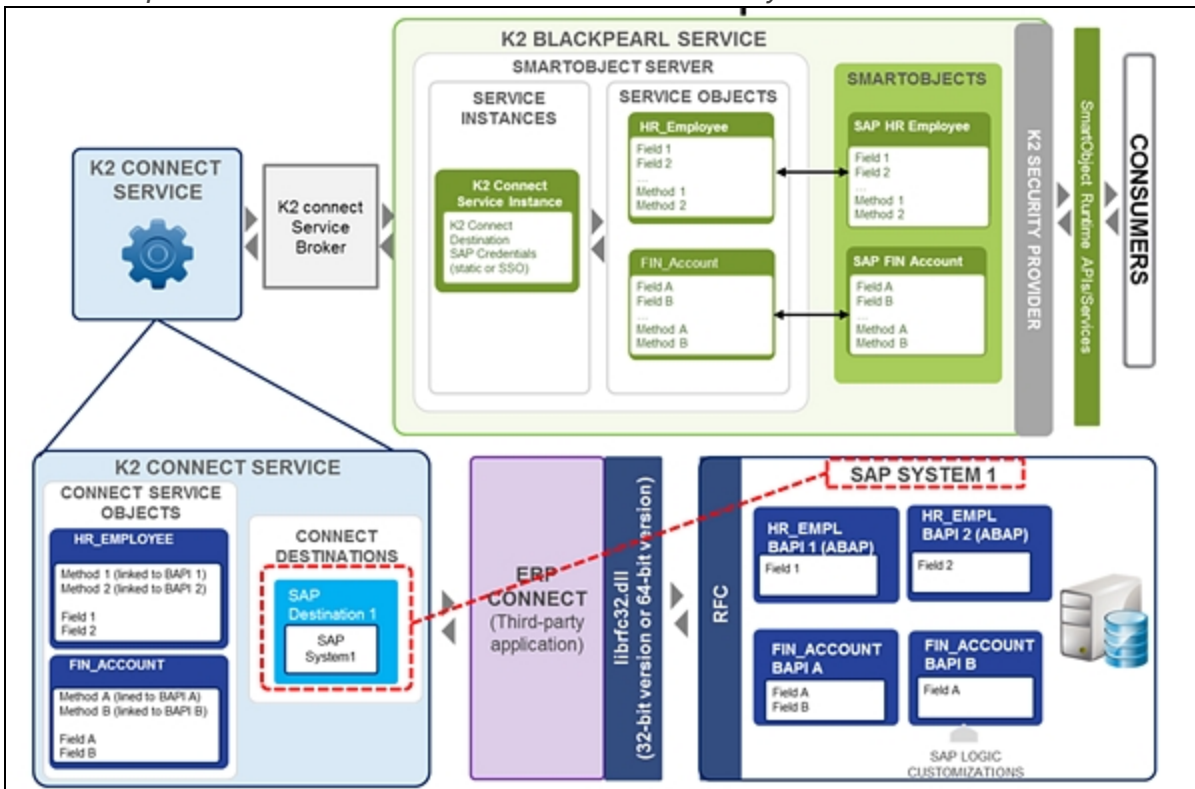
K2 connect uses a middleware component to communicate with a particular SAP environment. This middleware component is a third-party software package called ERPConnect from Theobald Software GmbH, and is used by connect to interact with SAP BAPIs via RFC. (ERPConnect is included with your K2 connect license, but needs to be installed separately). Notice that ERPConnect, in turn, relies on a particular .dll file (librfc32.dll) to communicate with SAP BAPIs. This .dll is an assembly provided by SAP.

ERPConnect acts as middleware between K2 connect and SAP



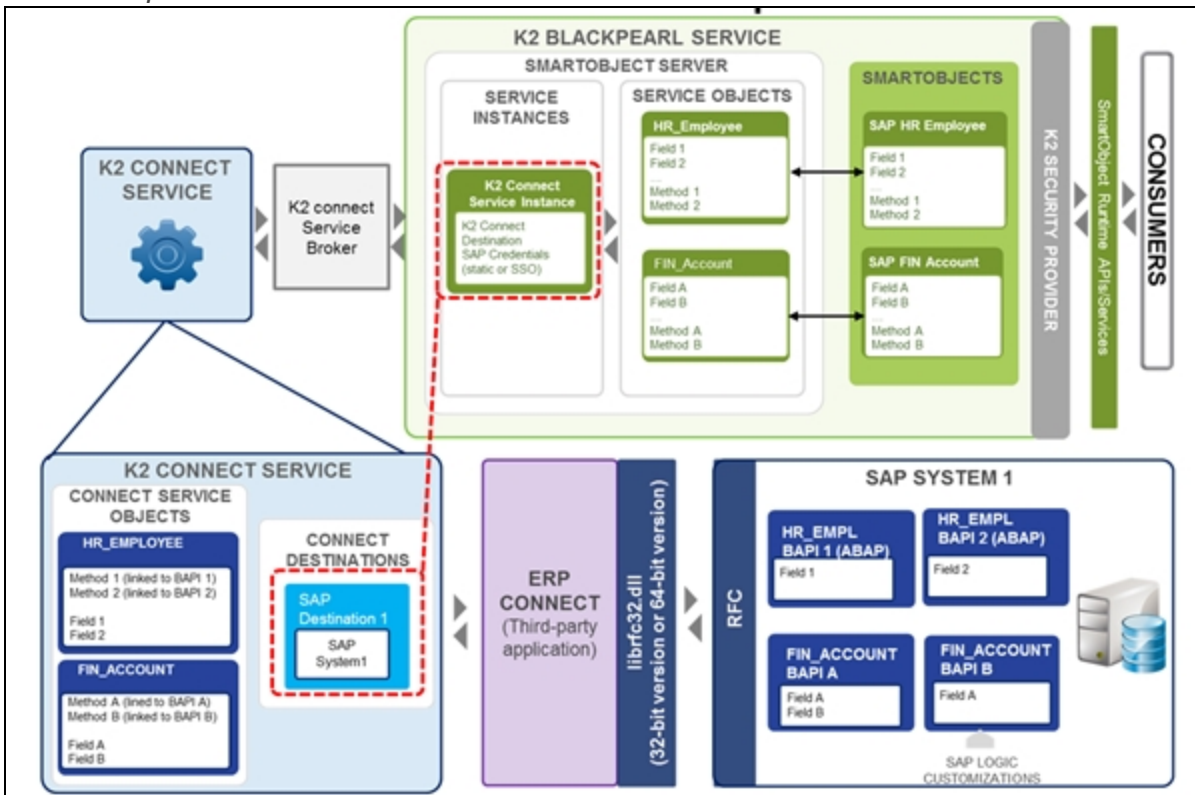
The next concept you need to understand is a Destination. K2 connect needs to store configuration settings so that it knows how to locate and interact with a particular SAP environment. This concept is known as a Destination, and a Destination typically contains SAP connection details such as Server Name, SAP system, and (perhaps) a username and password. A connect server would require at least one SAP Destination, but could have multiple Destinations that point to different SAP systems.

Relationship between a K2 connect Destination and a SAP System

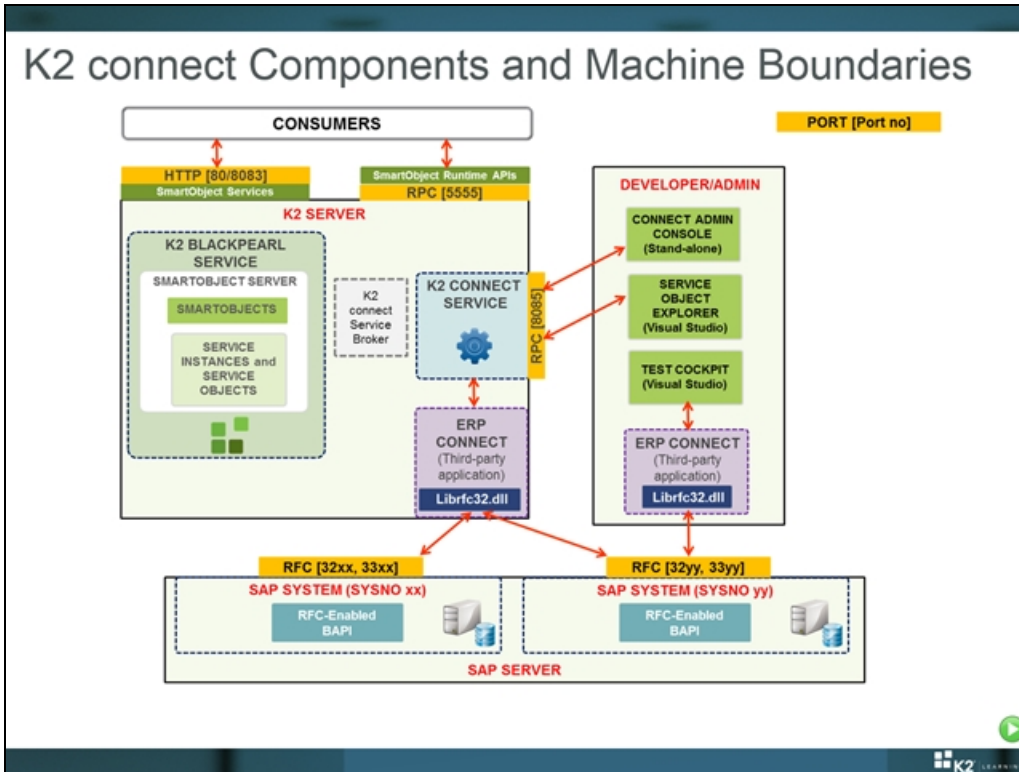


In most environments, each Destination configured on a K2 connect server has an associated K2 connect Service Instance on the K2 blackpearl server. The service instance points to a specific Destination, which in turns points to a specific SAP system, and that is how a K2 SmartObject ultimately "knows" which SAP system to interact with.

Relationship between a K2 connect Destination and the associated K2 connect Service Instance



K2 connect Components and Machine Boundaries



Before we talk about installing K2 connect, let's review the components of K2 connect and where these components need to be installed. Refer to the diagram in the slide for this topic (a larger version of this diagram is on the following page).

K2 Server

A physical server hosts the K2 blackpearl service as well as the K2 connect service. The K2 blackpearl service takes care of all SmartObject processing, while the K2 connect service handles interaction with SAP systems and handles the processing of connect Service Objects.

If you are familiar with SmartObjects, you should already know that the SmartObject server will take care of all the processing required to expose K2 SmartObjects to consuming applications, and that the consumers can access SmartObjects with the runtime .dlls (which use port 5555 to communicate with K2) or SmartObject services (which are exposed through HTTP ports on the K2 server by default).

When using K2 connect, the K2 connect service must also be installed on the K2 application server. (Note that K2 connect runs as a separate windows service). This service exposes itself via port 8085 to other connect components such as the developer and administration tools.

Notice that K2 connect relies on the third-party application called ERPConnect to communicate with SAP BAPIs. We will discuss this requirement more in the next topic.

Developer/Administrator workstation

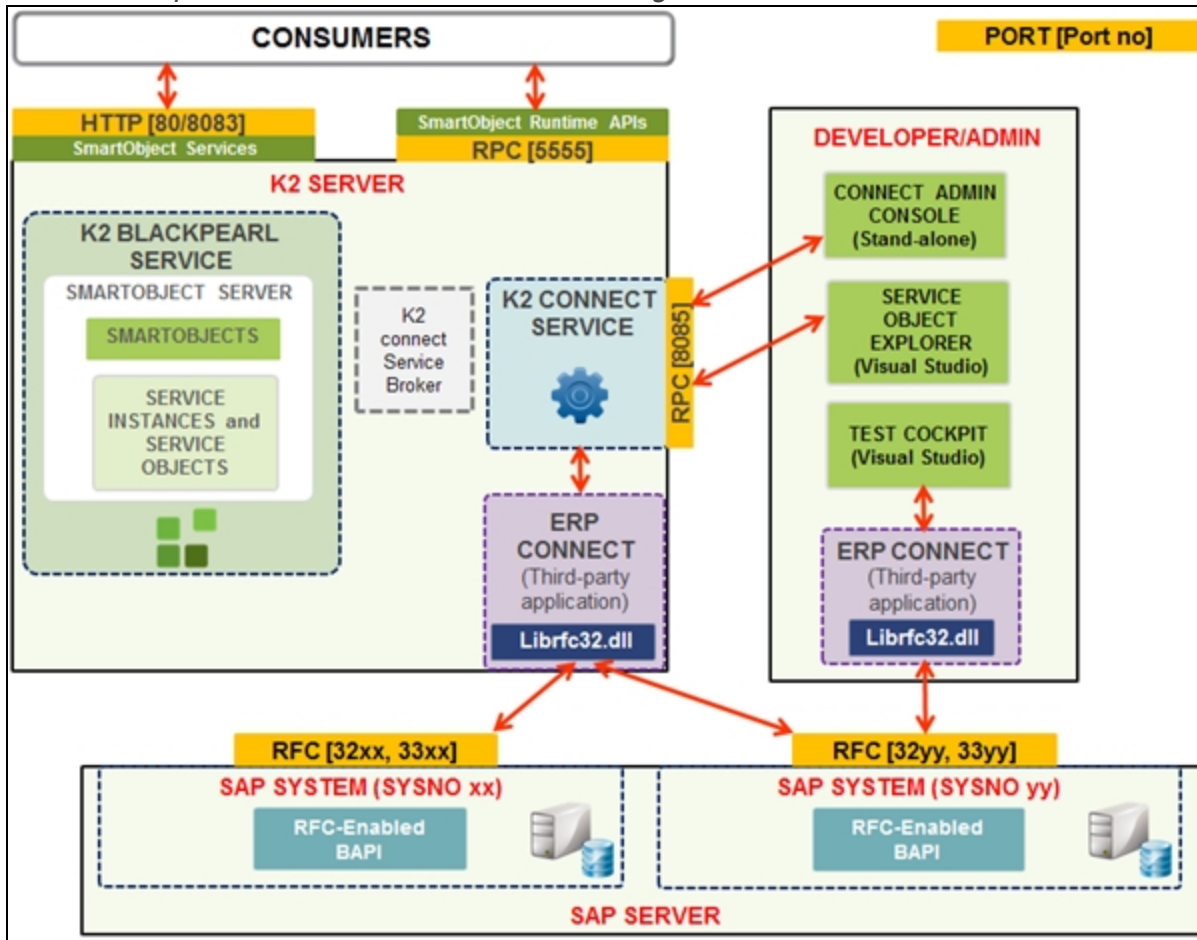
The developer installation will install the Visual Studio plugins (Service Object Explorer and Test Cockpit) and the K2 connect administration console. If needed, you can also install the administration console separately for administrator machines. While these components primarily communicate with the K2 connect service over port 8085, notice that the Test Cockpit communicates directly with SAP systems using the third-party tool ERPConnect. This means that you need to install ERPConnect on the developer workstations as well.

SAP Server

SAP environments could contain multiple SAP Systems. While it is not necessary to install any K2 or ERPConnect components on the SAP systems, note that ERPConnect will use a system-specific port to communicate with the SAP systems. In our sample diagram, we have two SAP systems (xx and yy). ERPConnect will use the SAP system number to

determine which port to use when connecting to the SAP system. You may need to open additional ports on firewalls so that ERPConnect can interact with the SAP RFC BAPIs. Also note that K2 connect can only interact with RFC-enabled SAP BAPIs.

Connect components and machine boundaries - diagram



K2 connect: Installation

K2 connect: Installation

- K2 blackpearl must be installed
 - Server components for server
 - K2 for Visual Studio for developer workstations
- Install lib32rfc.dll provided by SAP
 - Cannot be distributed by K2, available from SAP Service Marketplace
 - Must install the correct version (32-bit or 64-bit)
 - Required on each server hosting the K2 connect service
 - Required on developer workstations where Visual Studio plugins will be installed
- Install ERPConnect
 - Available for download from K2 portal
 - Separate installation from K2 connect
 - Redistribution license and installation permits use for K2 connect ONLY
 - Required on each server hosting the K2 connect service
 - Required on developer workstations where Visual Studio plugins will be installed
- Install K2 connect
 - Server installation for all K2 servers (license key required)
 - Client-side installation for developers/admins (no license required)

There are several prerequisites which must be fulfilled before you can install K2 connect. Remember that connect relies on K2 blackpearl, so you must have K2 blackpearl installed (blackpearl Server for server installations and K2 for Visual Studio for developer installations) before continuing. You will probably install K2 connect on multiple machines since there are server components and client components, therefore we recommend that you review the K2 Connect Planning Guide and K2 Connect Installation topics in the K2 connect product documentation before getting started, since your environment may have specific requirements.

Librfc32.dll

The next step is to install the lib32rfc.dll assembly, which is provided by SAP and required by the ERPConnect software. This assembly may not be distributed by K2 and you will require a valid SAP license to obtain the assembly. You can obtain the lib32rfc.dll from the SAP service marketplace at <http://service.sap.com/swdc> or a windows machine where the SAP GUI is installed.

Note

There are two different versions of the librfc32.dll: one for 32-bit systems and one for 64-bit systems: each is installed in a slightly different manner. We recommend you keep a copy of both versions, since the assembly is required for all K2 connect servers and all client machines where the K2 connect administration or developer tools will be installed. (You may also need to install msvc71.dll and msvc71.dll).

Check the K2 connect quick-install guide at <http://help.k2.com/en/KB000707.aspx> and the K2 connect documentation for more details

ERPConnect

You need to install the ERPConnect middleware on each server where K2 connect server will be installed, as well as each client machine where the K2 connect administration tools and K2 connect developer tools are installed. ERPConnect is a separate download and installation from K2 connect, and you may obtain it from the K2 portal at <https://portal.k2.com/downloads/k2connect>.

Note

The version of ERPConnect provided with K2 connect is a licensed redistributable version and the license

agreement allows the ERPConnect installation to be used with K2 connect ONLY. You may not use this version of ERPConnect to interact with SAP outside of K2 connect.

K2 connect

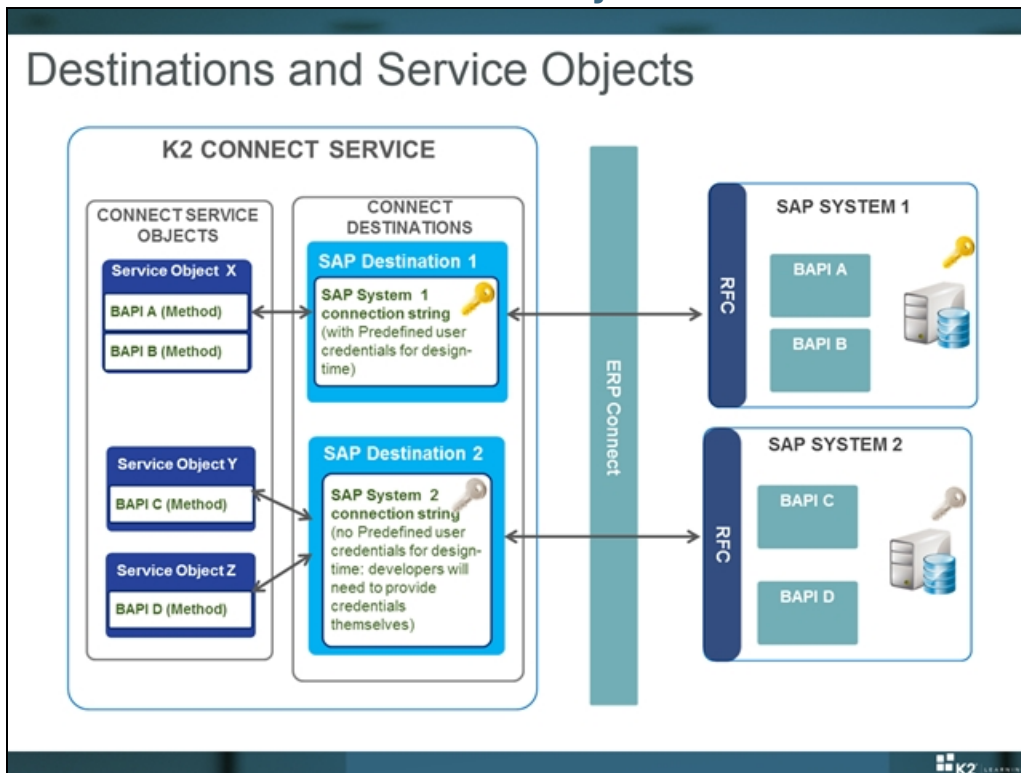
Once you have installed the prerequisites, you are ready to install K2 connect. Note that you will use the same installer to install both the server-side and client-side components of K2 connect, but only the K2 connect server requires a license.

Note

For more information on installing K2 connect, please refer to the K2 connect product documentation (available online at <http://help.k2.com/en/connectforsap.aspx>) and the quick-install KB article at <http://help.k2.com/en/KB000707.aspx>

After installing K2 connect, you will be prompted to configure a Destination. The next few topics will describe Destinations in much more detail.

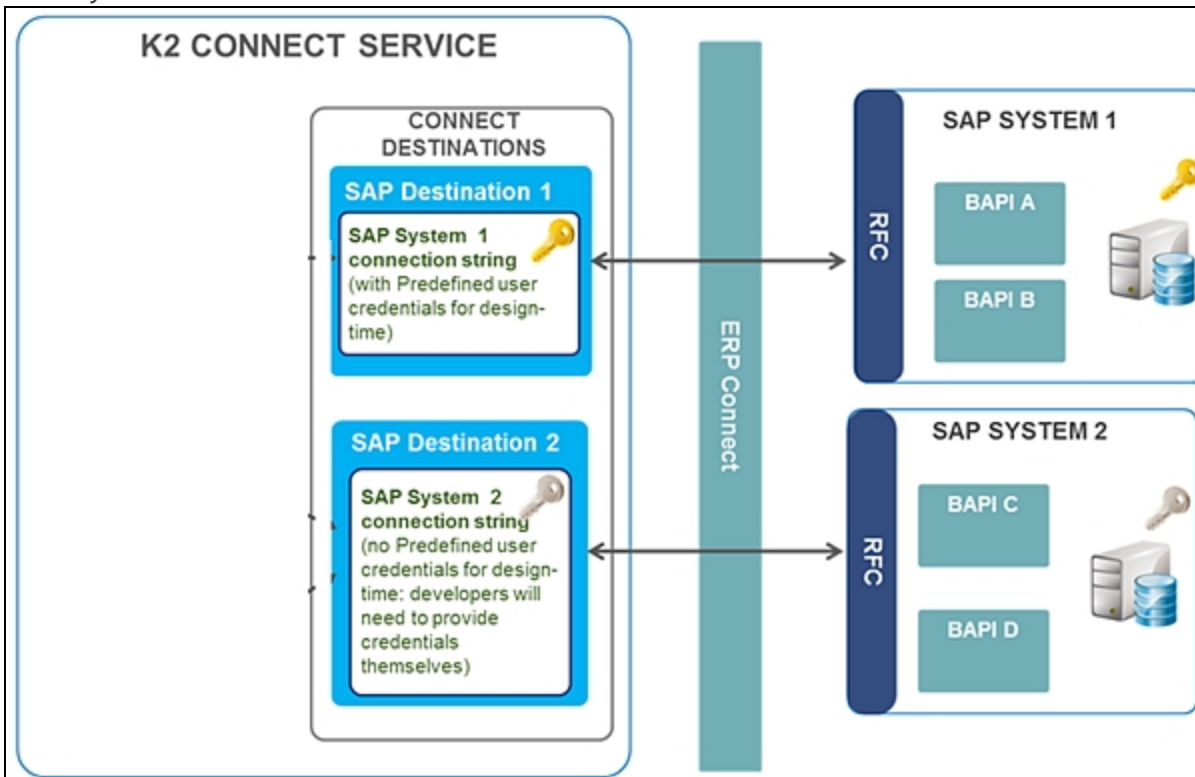
Destinations and Service Objects



Connect Destinations

Essentially, a K2 connect Destination is a configuration setting which tells the K2 connect server how to connect to a particular SAP system. A K2 connect server would require at least one Destination to be useful, but installations of K2 connect in larger enterprises would likely have multiple Destinations.

Consider the diagram in the slide for this topic. In this environment, we have two SAP systems (**SAP SYSTEM 1** and **SAP SYSTEM 2**), each of which contains two BAPIs which we will ultimately expose as K2 SmartObjects. The first task is to configure a connect Destination for each SAP System.



As part of the Destination configuration, you may include statically-defined credentials (username and password). These credentials are used at design time to discover the available BAPIs in the SAP system, and might be used by the K2 Service Instance at runtime. (More on that later)

In our example, **SAP SYSTEM 1** has been set up with a Predefined, static set of credentials while **SAP SYSTEM 2** has been set up without credentials. (This means that whenever a developer wants to explore the **SAP SYSTEM 2** they will have to provide credentials.)

Once the configuration details are entered, each SAP system is exposed as a connect Destination (**SAP Destination 1** and **SAP Destination 2**, respectively).

The connection string for a SAP Destination usually looks something like this (not all values are always required. Check with your SAP administrator what the connection string should be for your target SAP system).

```
Code
ashost=[IP Address or Hostname] sysnr=[NR] client=[NR] lang=[language] user=[USERNAME] passwd=[PASSWORD]
```

Here is an example of a connection string for a hypothetical SAP system, with statically-defined credentials:

```
Code
ashost=sapdev.denallix.com sysnr=KR3 client=900 lang=EN user=DEV passwd=sapdeveloper
```

Once you configure a Destination, you can Test it to make sure that K2 connect can locate and connect to the targeted SAP system. Destinations are stored in the K2 connect database so that they can be imported by other K2 connect Servers in the same farm.

Service Objects

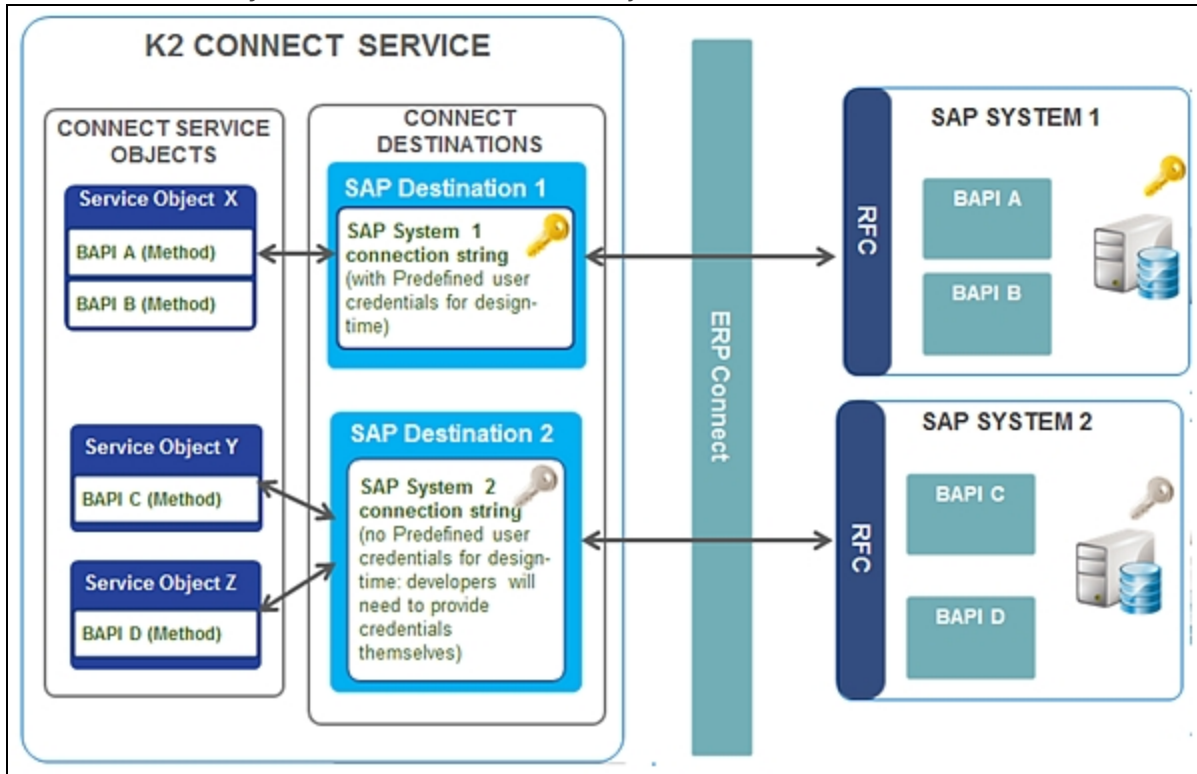
Once a Destination is set up, developers can start creating connect Service Objects which utilize the available BAPIs in the targeted SAP systems.

Consider the diagram below. Here, developers have created three connect Service Objects (**Service Object X**, **Service Object Y** and **Service Object Z**). Service Objects may contain one or more Methods which are typically (but not

necessarily) linked to SAP BAPIs. Service Objects could contain multiple methods (for example, **Service Object X** contains two methods; one for each of the BAPIs available in **SAP SYSTEM 1**).

In most situations, multiple Service Objects would exist for a particular Destination (in our example, **Service Object Y** and **Service Object Z** both point to the same **SAP SYSTEM 2**, but the Service Objects use different SAP BAPIs).

connect Service Objects, Destinations and SAP Systems



Configuring Destinations

Configuring Destinations (1/2)

- After installing connect, use the administration tool to set up one or more Destinations
- Destination names should be simple and clear
- Optional: specify static design-time credentials for the SAP system
 - If no credentials in connection string (a.k.a. **Predefined User Credentials**), developers will need to provide SAP credentials at **design time**
 - **Runtime** credentials are determined by the Service Instance associated with the destination. Could be static or SSO
- BAPI security and access is managed by SAP
 - Only RFC-enabled BAPIs will be available to developers
- Developers will use these destinations into the Visual Studio tooling

Normally, administrators configure one or more Destinations to SAP Systems using the K2 connect administration tools. Each Destination should be configured with a clear and descriptive name so that it is obvious which SAP environment the Destination targets.

As we mentioned in the previous topic, a Destination can, but doesn't have to include static, Predefined credentials. If no credentials have been provided, developers will need to enter them at design time before they will be able to browse the targeted SAP system (developers can always provide their own credentials, even if static credentials were not provided). Runtime credentials are determined by the K2 Service Instance associated with the Destination, and could use static credentials as well or use a Single Sign-On (SSO) approach for dynamic, user-specific credentials. (More on that later)

Note

At design time, the list of BAPIs available to developers is limited to RFC-enabled BAPIs and could be trimmed by SAP security as well.

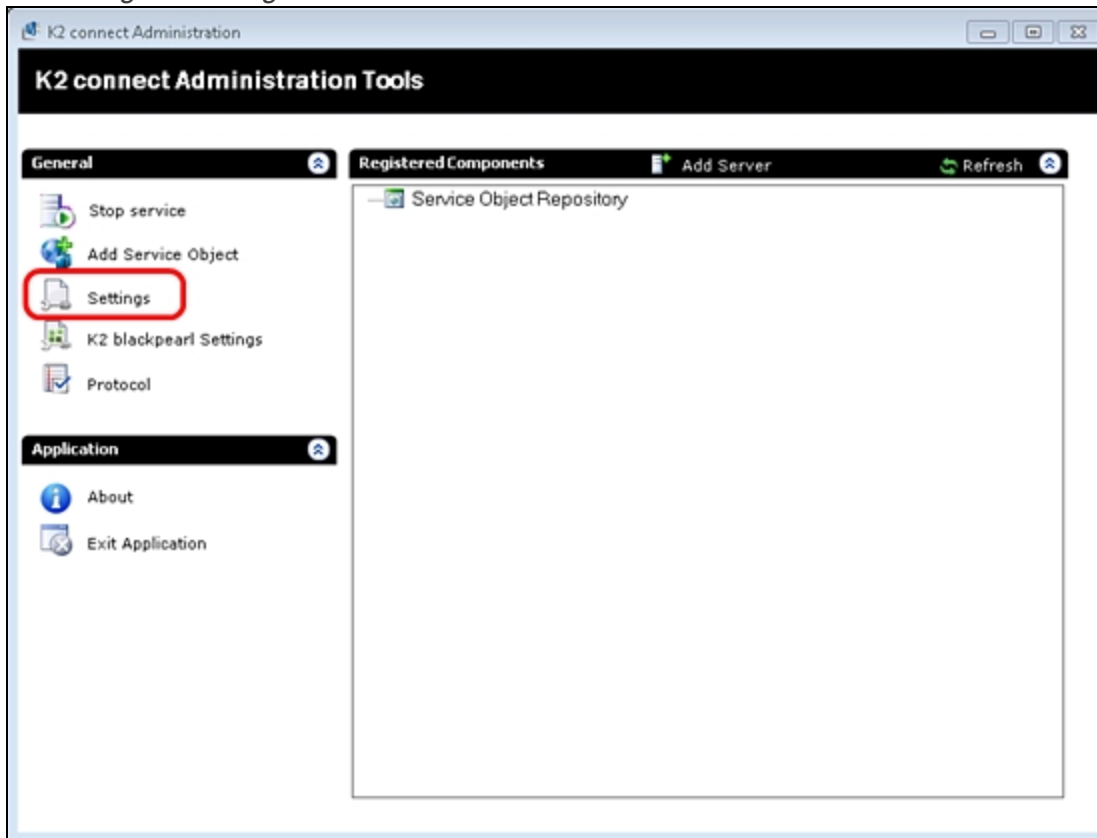
If your organization uses user-level security for SAP BAPIs, do not include credentials in the Destination configuration. This forces developers to enter credentials at design time, and you would use a Single Sign-On (SSO) approach for runtime credentials. This will help to ensure that the security defined in the SAP environment is maintained whenever users interact with the SAP BAPIs.

If you have a non-sensitive SAP system where user-level security is not critical, static credentials are a simpler option since it is slightly easier to configure and troubleshoot.

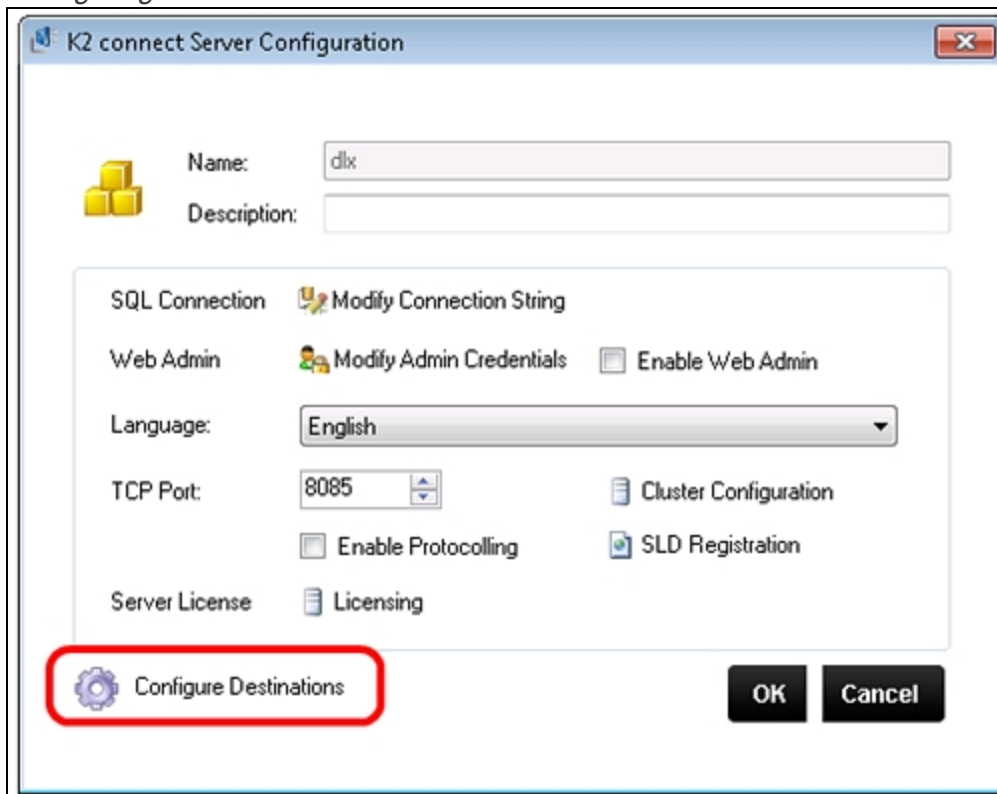
Once you have configured Destinations, developers can connect to the K2 connect server and use the Destinations in their Visual Studio environments to explore the available BAPIs, and use those BAPIs in their connect Service Objects.

The series of screenshots below illustrates how you would set up a Destination using the K2 connect administration console.

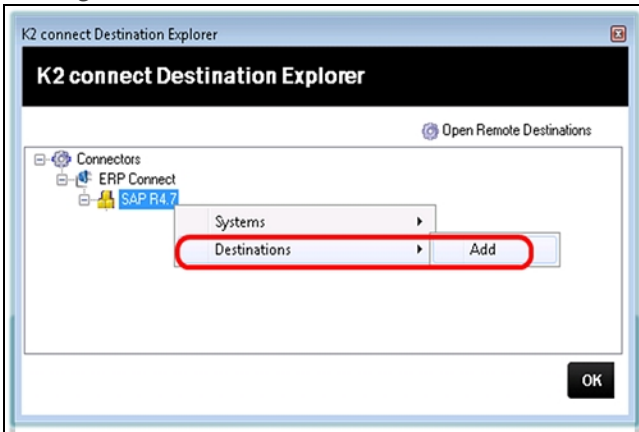
Accessing the Settings



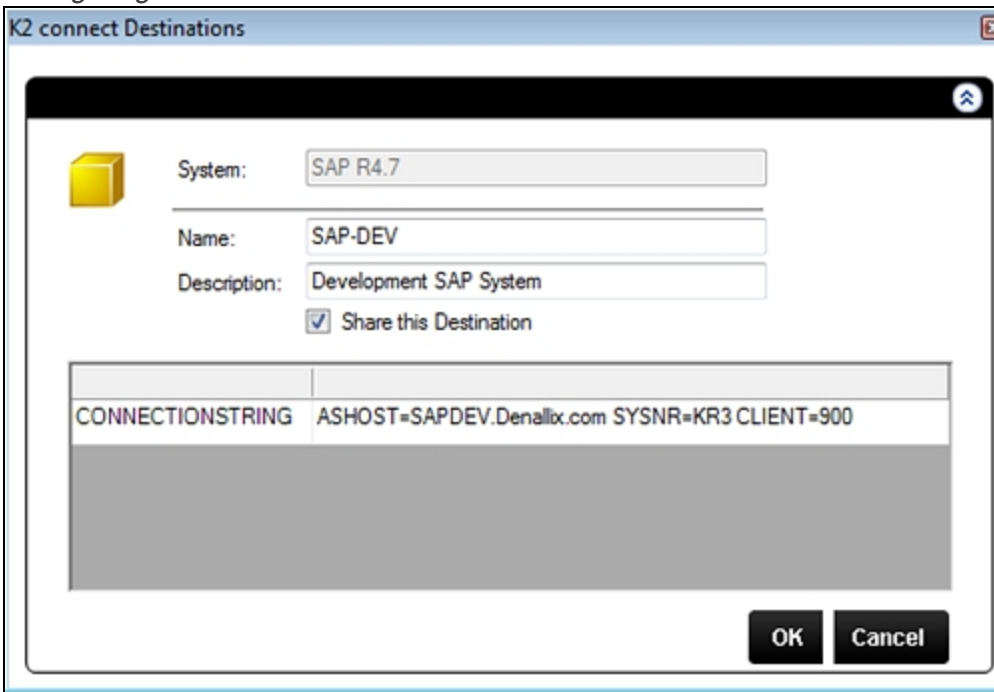
Configuring Destinations for the K2 connect Server



Adding a new Destination for the ERPConnect adapter



Configuring the Destination



Note

The **Share this Destination** setting will make the destination discoverable so that other servers can import the Destination. It is mostly used to import Destinations for NLB/Farm environments). We recommend enabling this setting for NLB environments.

Once you add a Destination, K2 connect creates a Service Instance for the Destination behind the scenes. The Destination Name is sent to the **Extra** setting for the Service Instance and is used by the Service Instance Service Objects to determine which K2 connect Destination they need to use.

Connect Destination and its associated Service Instance

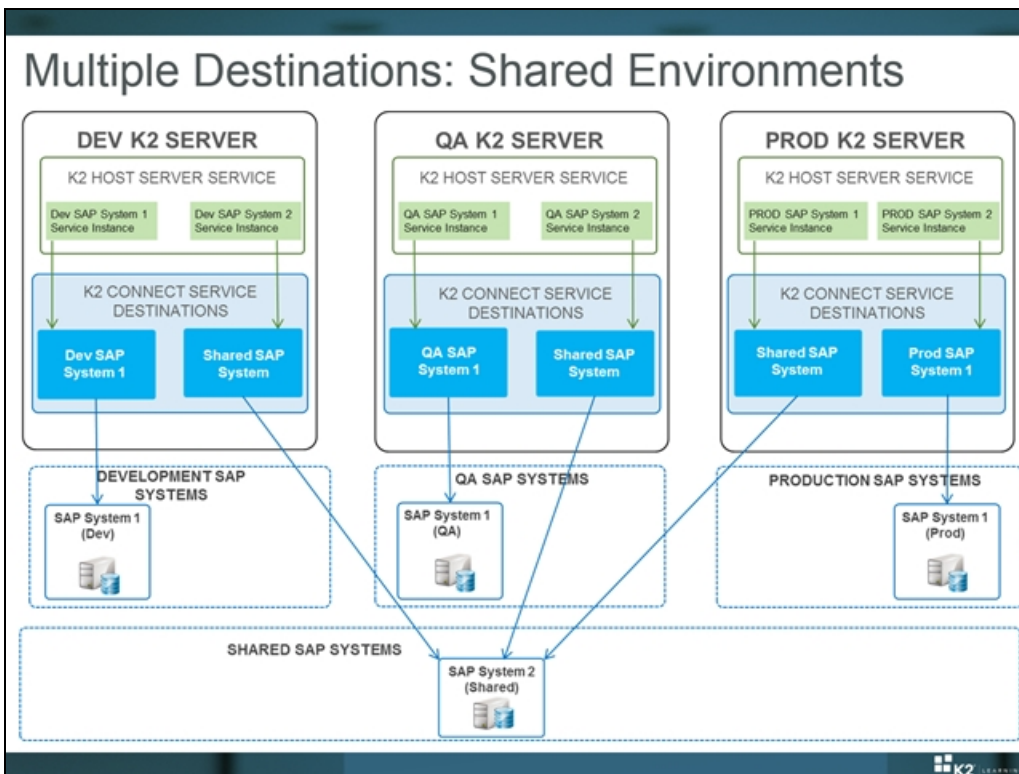
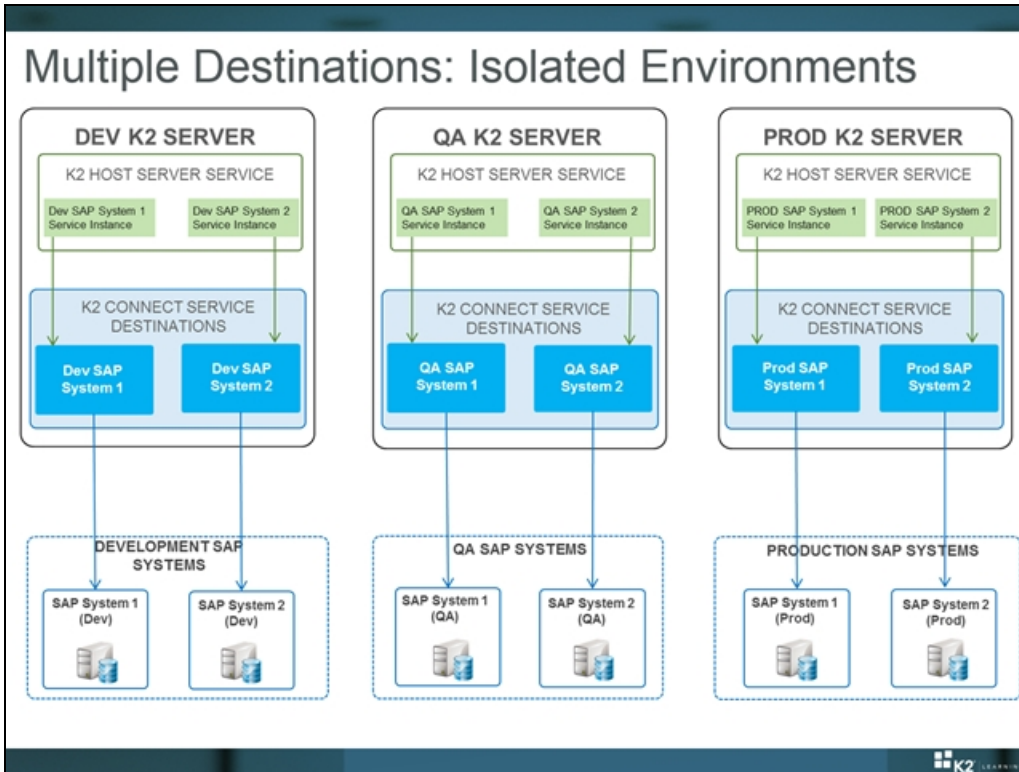
The screenshot displays the SAP K2 connect interface with two overlapping windows. The background window, titled "K2 connect Destinations", shows a destination configuration for "SAP R4.7". The "Name" field is set to "SAP-DEV" and the "Description" is "Development SAP System". A "CONNECTIONSTRING" field contains the value "ASHOST=SAPDEV.Denallix.com SYSNR=KR3 CLIENT=900".

The foreground window, titled "Add Service Instance", is used for configuring service authentication. It includes the following fields and options:

- Service Authentication:**
 - Authentication Mode: Static
 - Impersonate:
 - IsRequired:
 - Enforce Impersonation:
- Security Provider:** (Dropdown menu)
- User Name:** sapdev
- Password:** (Text field) with "Retain Existing Password" checked.
- Extra:** SAP-DEV (This field is circled in red, and a red arrow points from it to the "Name" field in the background window, with the label "SAP Destination Name" next to the arrow).
- Service Keys:**
 - server - Required: dx
 - port - Required: 9085
 - sapso - Required: false
 - typemapping: (Text field)

At the bottom of the "Add Service Instance" window are "OK" and "Cancel" buttons.

Multiple Destinations



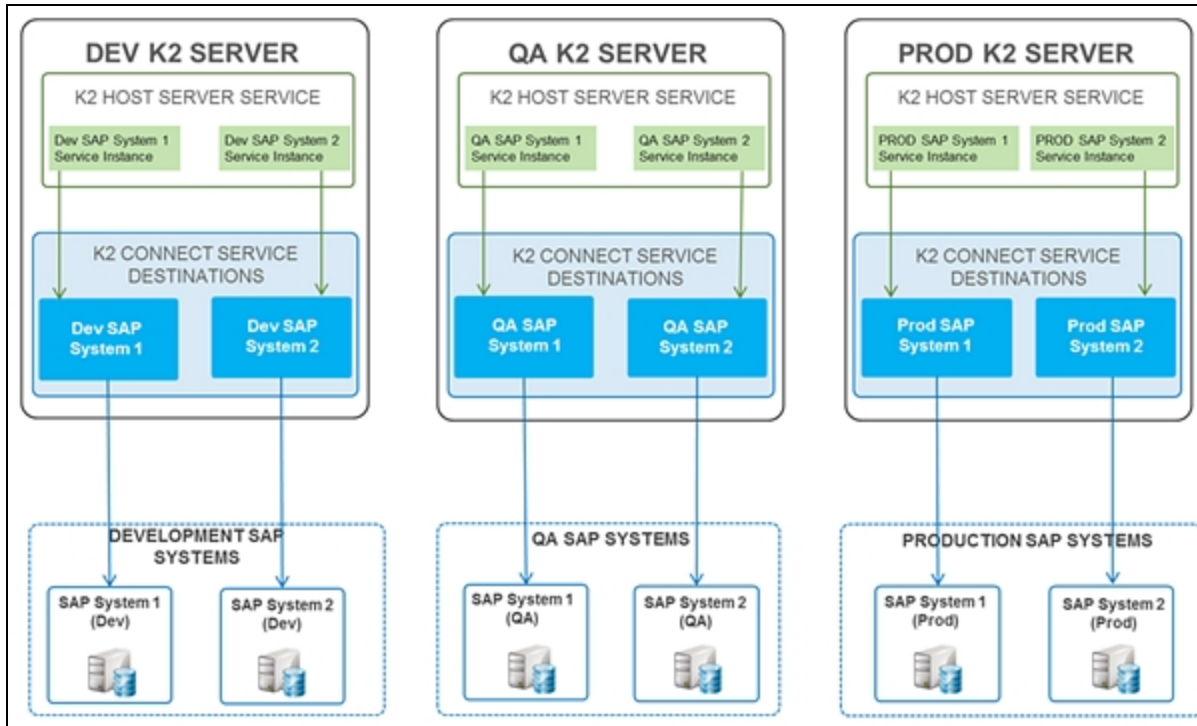
Larger enterprises could have multiple SAP Systems, and these systems could be dedicated to specific environments (such as Development and QA) or shared between environments (such a Systems containing common lookup information that is used across all environments).

When you configure Destinations for multiple K2 environments, you should understand the scope of the target SAP environment. SAP Systems might be completely isolated or shared between environments, and your connect Destinations should be configured appropriately.

Isolated SAP Systems and Destinations

Consider the diagram below. It illustrates three hypothetical Environments (**DEV**, **QA** and **PROD**). Each environment has a dedicated K2 server.

Isolated Environments and Destinations

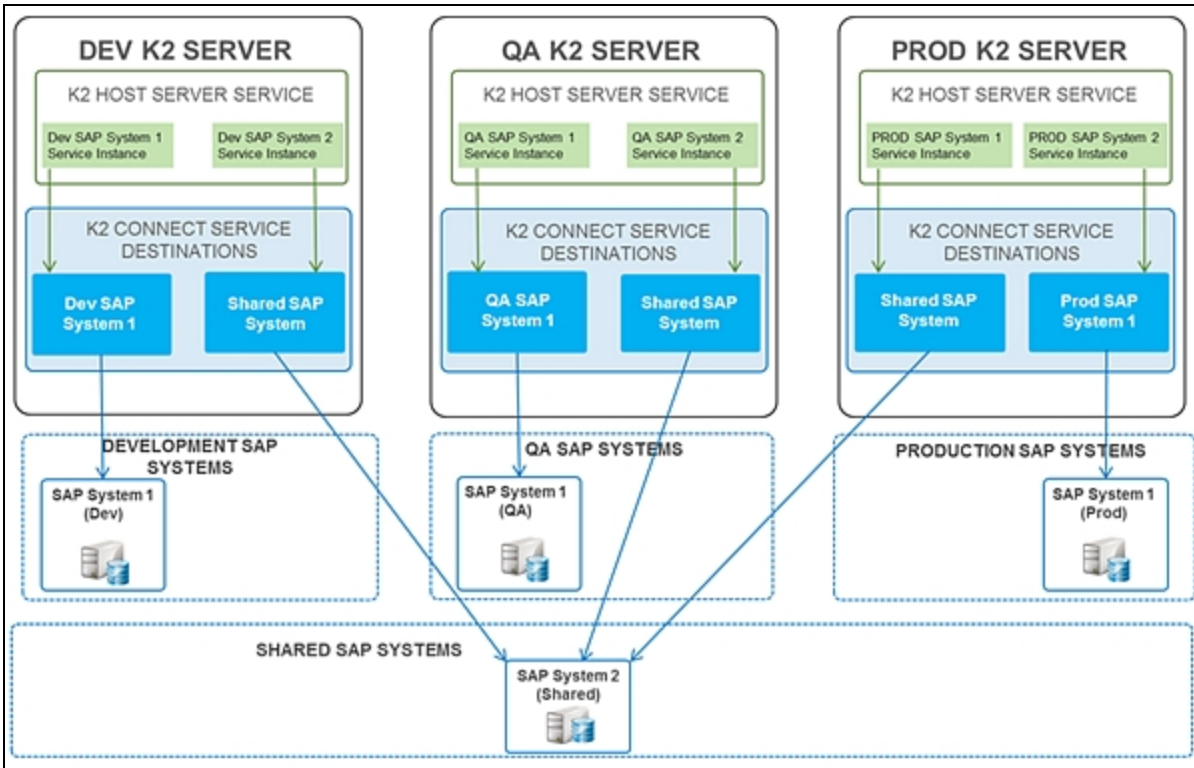


Suppose that each Environment has two SAP systems, and the SAP systems are completely isolated from each other. When configuring Destinations for the SAP systems, the connect Administrator would therefore create separate Destinations per K2 server for each environment. The diagram illustrates this concept.

Shared SAP Systems and Destinations

It is also possible to share a SAP system between K2 servers. Consider the diagram below. Here, we have the same three hypothetical environments (**DEV**, **QA** and **PROD**). Each environment has one SAP system which is specific to that environment and there is a common, shared SAP environment which is accessed by all the K2 environments. In this case, each K2 server has a second destination which points to the same, shared SAP system, as you can see in the diagram.

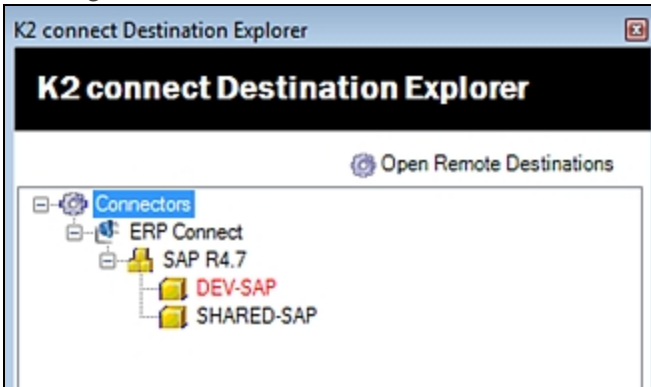
Shared Environments and Destinations



Destination Naming

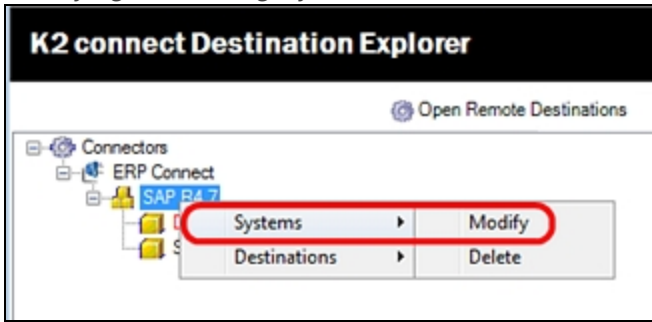
Since it's easy to get confused when working with multiple SAP environments, it is a good idea to give each environment a simple, obvious and descriptive name. Consider the screenshot below, which illustrates two Destinations configured by the connect Administrator.

Naming Destinations

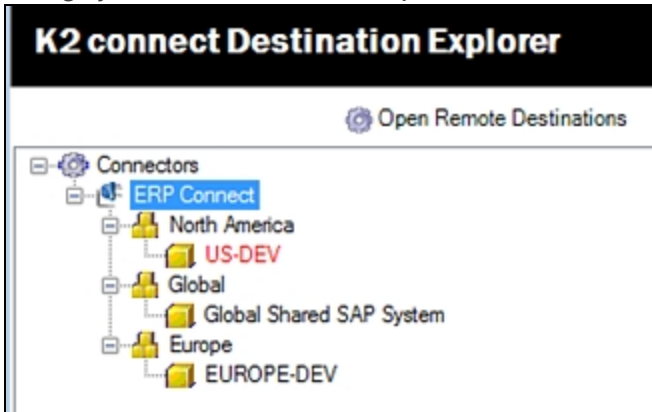


In even larger environments, you could consider implementing a tree-view structure by renaming the System node, as shown below. This can make it even easier for developers to locate and select the correct SAP system when they create their Service Objects. The screenshots below illustrate this approach, where we have configured Systems based on geographic usage.

Modifying and adding Systems



Using systems to structure multiple SAP environments

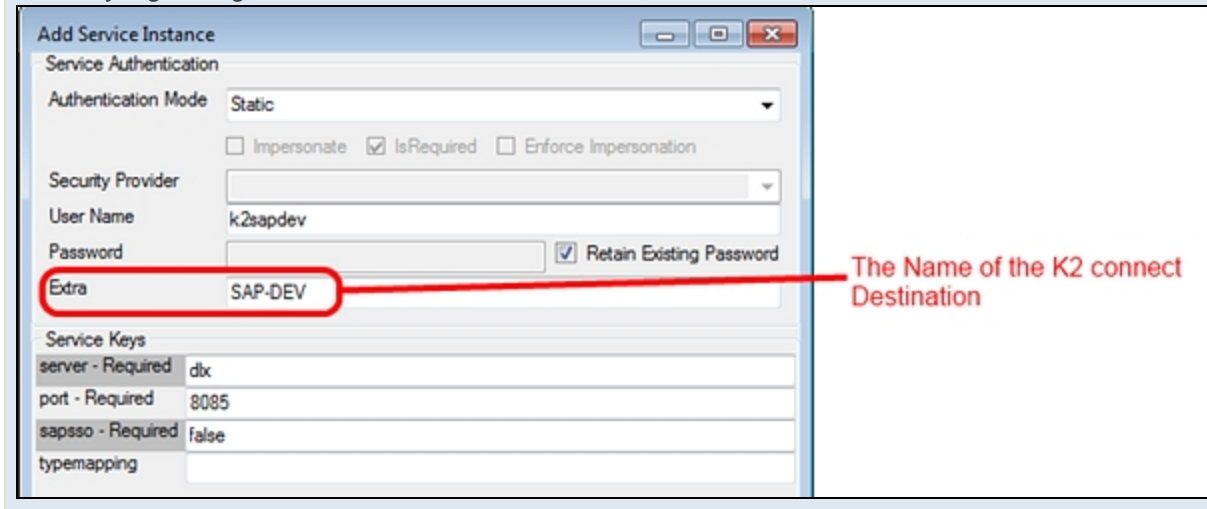


Note

At the time of writing, the K2 connect administration console only supports a single Destination > Service Instance configuration.

If you are configuring multiple Destinations, you will need to manually register Service Instances for each SAP Destination, and to set the Extra property of the Service instance to the name of the K2 connect Destination.

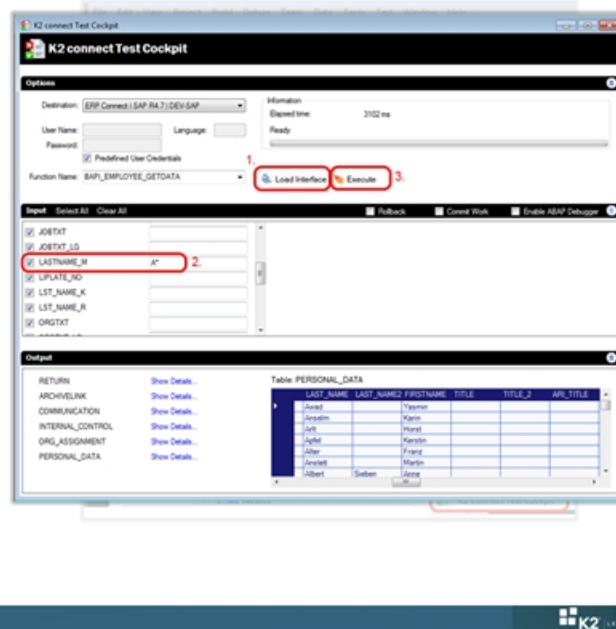
Manually registering a Service Instance for the K2 connect Service Broker



Service Object Explorer and Test Cockpit

Service Object Explorer and Test Cockpit

- Visual Studio plugins used to explore and test SAP BAPIs
- Connect to a K2 connect server and open destinations
- Use Service Object Designer to explore available SAP BAPIs and launch Test Cockpit
- Use Test Cockpit to test SAP BAPIs

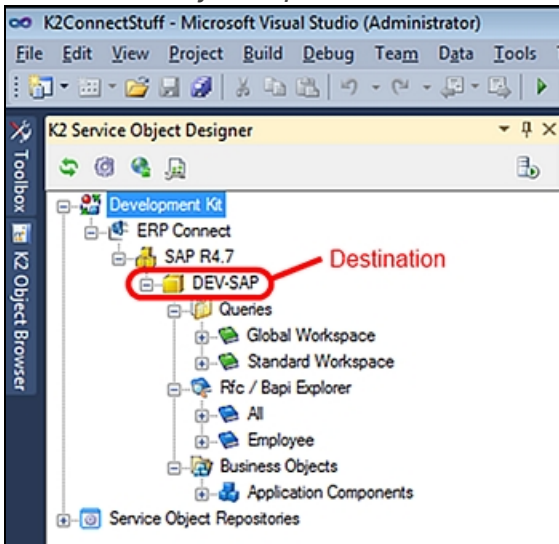


After you have configured K2 connect destinations, you can use the Service Object Explorer and Test Cockpit tools in Visual Studio to test the connection to the SAP system. Note that these tools are Visual Studio plugins, normally installed in a developer environment. If your connect administrator cannot have Visual Studio installed, a developer will need to perform the tests with the Visual Studio tools.

Service Object Designer

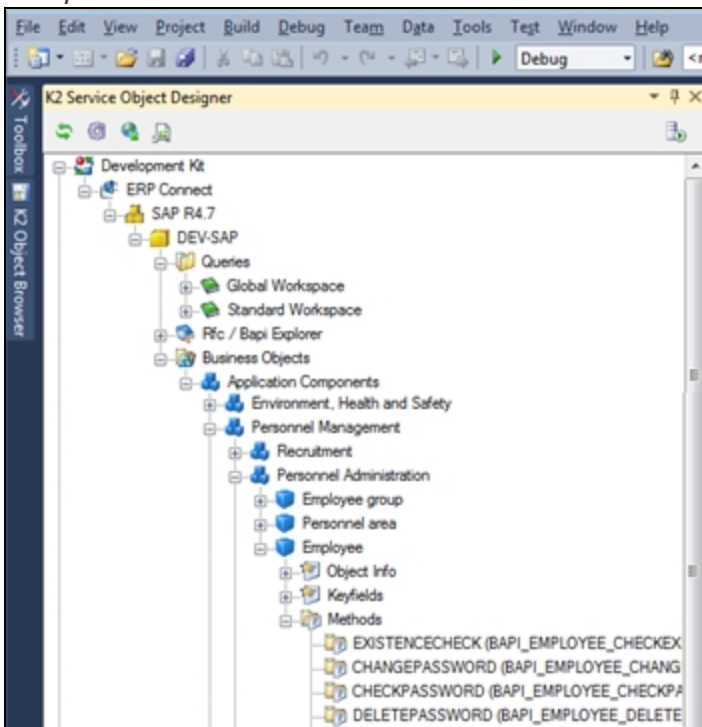
The Service Object designer is used to connect to SAP systems defined as Destinations in K2 connect, and browse the available BAPIs. (You can also use it to manage the Service Object repositories, but we will cover that functionality in the module **Developing with K2 connect**). You will configure the Service Object Explorer to connect to a K2 connect server. During this process, the Destinations defined for the service are displayed and the developer can explore the available BAPIs. If multiple Destinations are configured, they all will appear in the list, so it is important to give the Destinations obvious names.

The Service Object Explorer in Visual Studio



The Service Object Explorer exposes a tree-like navigation structure where you can browse the available BAPIs, as shown in the sample below.

Sample BAPI treeview

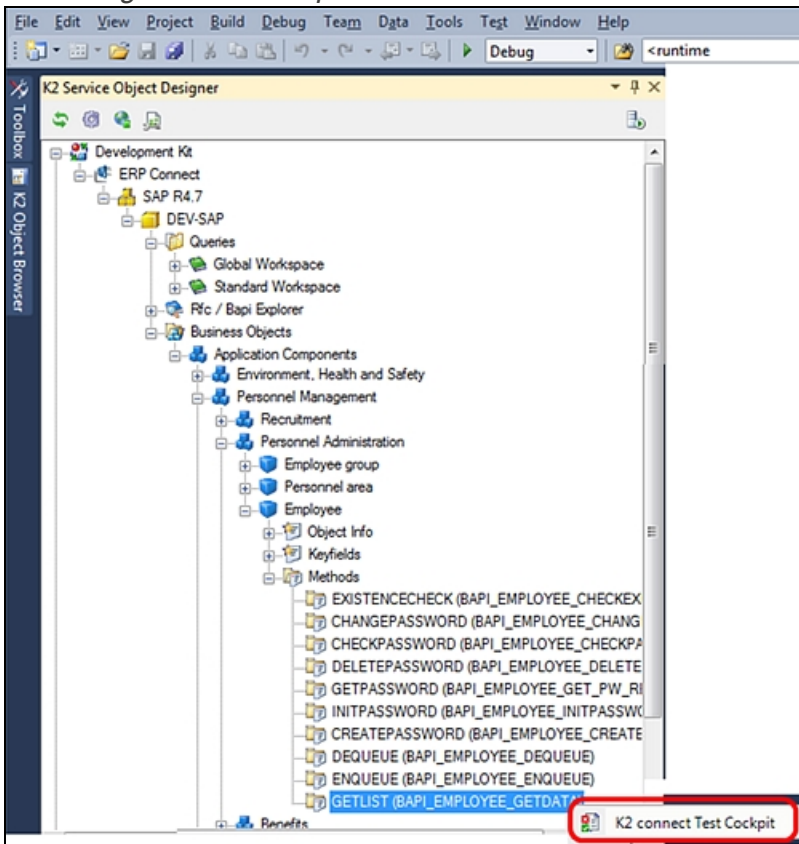


The Test Cockpit

You can use the Test Cockpit to execute a particular BAPI without committing data. This function is useful if you want to test the operation of the target SAP system, or if you want to explore the BAPI in more detail.

To launch the Test Cockpit, locate the BAPI you want to test, right-click it and select K2 connect test Cockpit.

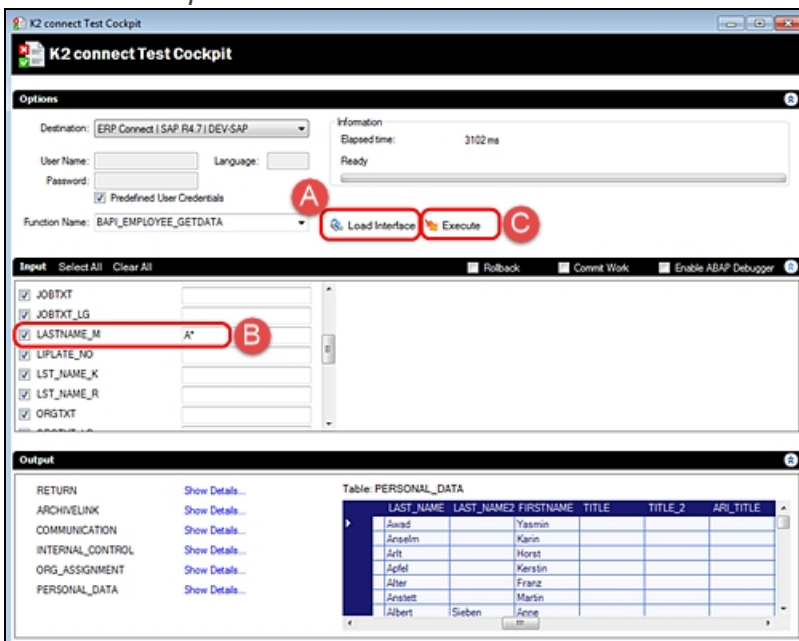
Launching the Test Cockpit



This will launch the Test Cockpit window. To test a BAPI, click the Load Interface button (A), which will populate the Input and Output structures for the BAPI. Enter the required and optional Input parameters (B) and then click the Execute (C) button to run the BAPI. The return results will depend on the BAPI, so click through the Output window to see a table of results for Output.

Note
If any errors are returned by SAP (such as missing parameters), they will likely be in the RETURN Output property.

The Test Cockpit



Design-time Authentication

Design-time Authentication

- Developers need permission to restart the K2 connect service
- Only RFC-enabled BAPIs will be visible to developers
- Authentication is defined by the Destination configuration
- Predefined credentials
 - Set by connect administrator
 - The list of available BAPIs are based on the static credentials
- No predefined credentials
 - Developer will need to provide SAP authentication credentials
 - The list of available BAPIs could be filtered based on the developer-provided credentials
- Design-time authentication is used for the Service Object Designer and Test Cockpit

K2 connect security is handled differently in Design-time vs. Runtime. (Design-time refers to the Service Object Designer and Test Cockpit tools in Visual Studio which developers use to create connect Service Objects, while Runtime refers to SmartObjects being executed by users and applications).

Let's look at some security considerations for design-time.

Developers need rights to restart the K2 connect service

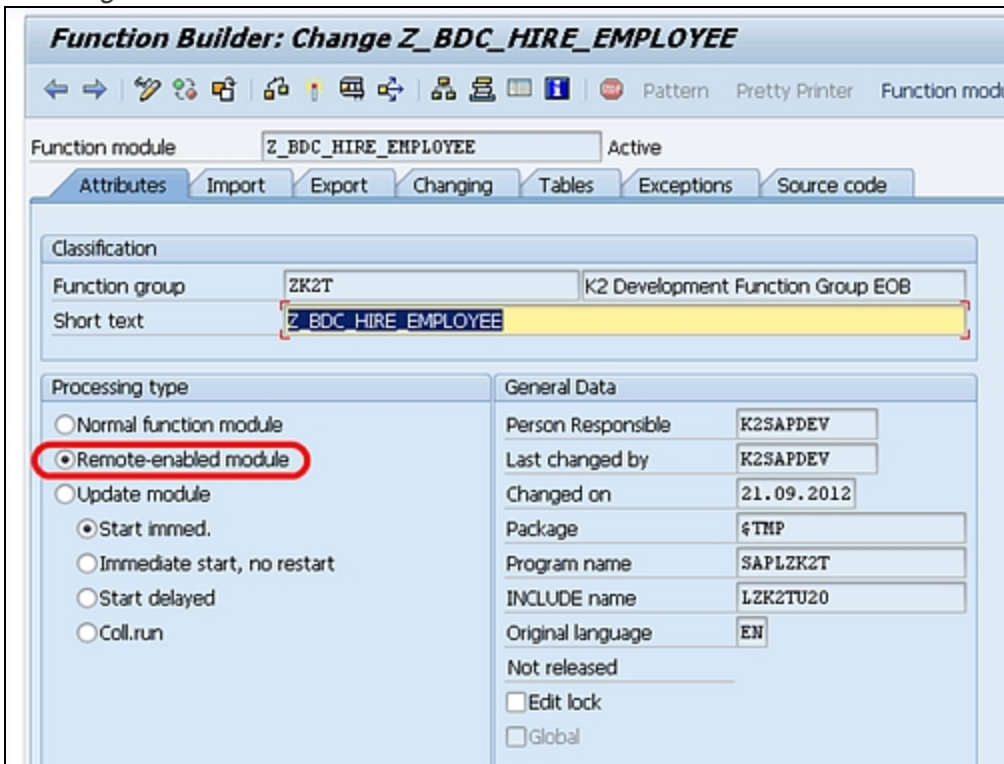
Developers will need permission to remotely restart the K2 connect service, because as part of the deployment process the K2 connect service may be restarted to load the updated definitions. By default the local administrators security group in Windows has this permission, but if you want to give developers the right without adding them to the local administrators group, see the following Microsoft KB article: <http://support.microsoft.com/kb/325349>

RFC-enabled BAPIs

Only RFC-enabled BAPIs will be available to developers. (We won't discuss the internal semantics of SAP in this training module, but essentially, BAPIs are RFC-enabled function modules on SAP. Regardless of the underlying SAP terminology, K2 connect uses the term BAPI to refer to these remote-enabled function modules.)

Your SAP administrator will know how to enable RFC for function modules in your specific environment. To register a remote function module in ABAP, the SAP administrator needs to register the module as remotely callable in the RFC server system, using the function module Administration screen (transaction code SE37). They should set the BAPI to be remote-enabled. (For more, refer to the SAP documentation. Screenshot of this setting in a SAP R/3 system shown below)

Enabling a BAPI for remote access



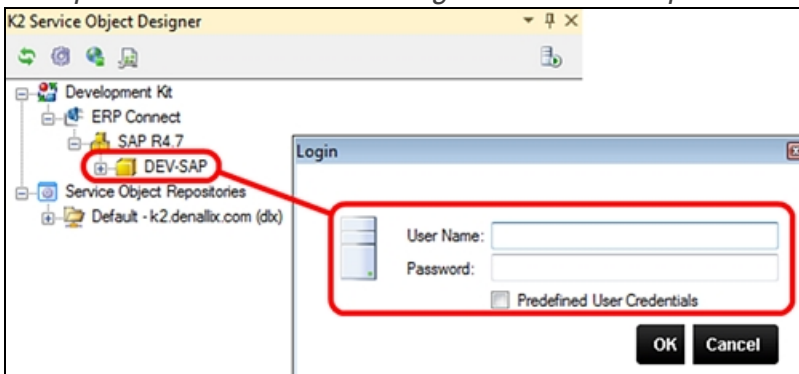
The screenshot shows the SAP Function Builder interface for the function module **Z_BDC_HIRE_EMPLOYEE**. The function module is set to **Active**. The **Classification** section shows the function group **ZK2T** (K2 Development Function Group EOB) and the short text **Z_BDC_HIRE_EMPLOYEE**. The **Processing type** section has the **Remote-enabled module** radio button selected and circled in red. The **General Data** section includes fields for Person Responsible (K2SAPDEV), Last changed by (K2SAPDEV), Changed on (21.09.2012), Package (\$TMP), Program name (SAPLZK2T), and INCLUDE name (LZK2TU20). There are also checkboxes for **Edit lock** and **Global**.

Design-time authentication

As we discussed in a previous topic, the connect Administrator may choose to include Predefined security credentials in the connect Destination connection string. If no security credentials were supplied in the connection string, developers will be prompted to enter SAP credentials whenever they explore the target SAP system with the Service Object Designer or the Test Cockpit tools.

If static credentials were used and the developer selects the option to use **Predefined User Credentials**, note that the list of BAPIs will be the same for all developers. If the developer provides credentials themselves, the list of available BAPIs may be filtered based on SAP security. Use the approach that best suits your organization's security requirements.

Prompt to enter credentials at design time. Note the option to use Predefined User Credentials



The screenshot shows the K2 Service Object Designer interface. A **Login** dialog box is open, prompting for **User Name** and **Password**. The **Predefined User Credentials** checkbox is checked and circled in red. The **DEV-SAP** destination is also circled in red in the background tree view.

Runtime Authentication

Runtime Authentication

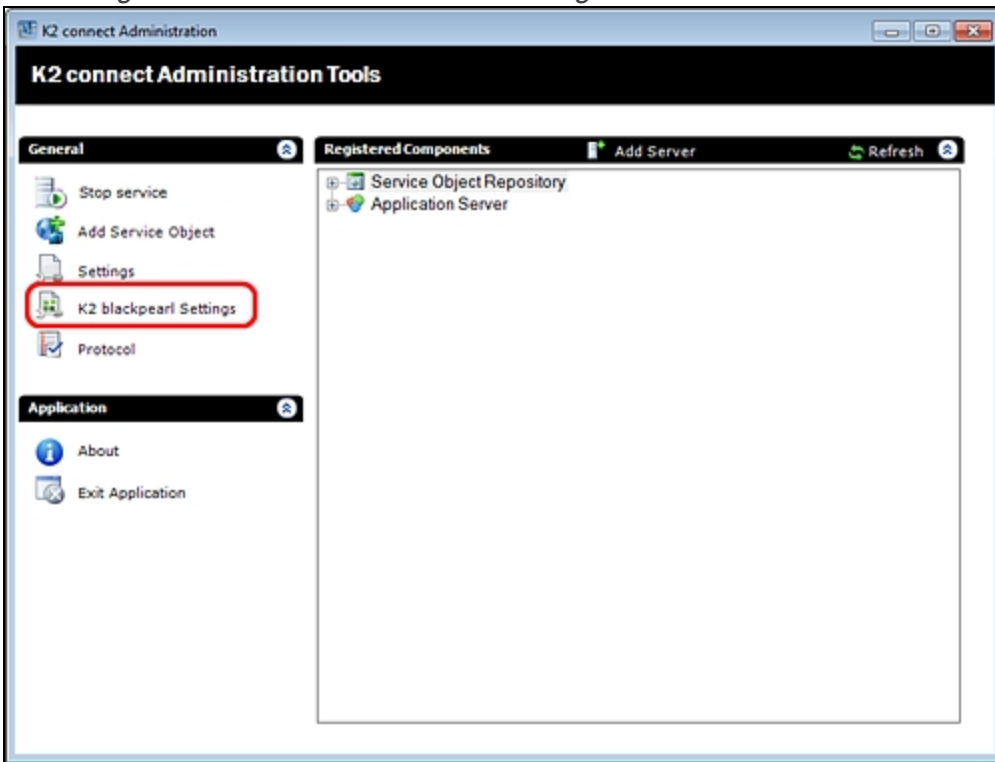
- Authentication is determined by the Service Instance Authentication configuration (e.g. static/SSO)
- Change the authentication setting with the connect Administration tool
- Static Credentials
 - Uses static credentials at runtime (therefore no user-level security)
- K2 SSO
 - Uses the K2-cached credentials for the current user
 - Based on the K2 SSO provider
 - User credentials are "converted" to equivalent SAP credentials
 - Credentials managed in K2 workspace
 - Can write code to add cached credentials
- SAP SSO
 - Requires a SAP Logon Ticket ("token") input parameter for each Service Object method call
 - Requires SAP Netweaver portal to generate the Logon Ticket

Runtime authentication is configured per Destination and the settings are applied to the Service Instance for the Destination. We recommend using the K2 connect administration tool to update the Runtime security for a destination, but you may need to update the security setting manually if multiple Destinations are configured.

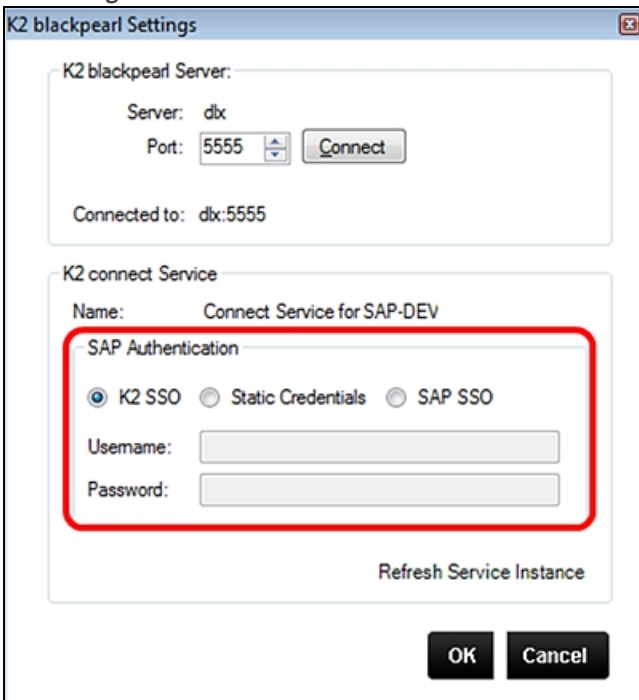
You have three approaches to runtime authentication for K2 connect: Static Credentials, K2's internal Single Sign On (SSO) provider or a SAP SSO mechanism based on SAP Logon Tickets.

To configure the runtime security for a connect Destination with the connect administration tool, click **K2 blackpearl settings** and then select the appropriate authentication mechanism from the radio buttons.

Accessing the connect Service Instance settings from connect Administration console



Selecting the runtime authentication mechanism



The three authentication options are described below.

Option	Behavior
Static Credentials	Enter a static SAP username and password. All communication between K2 and SAP will happen in the context of this account, so essentially all users will have the same level of access and SAP will not have context of the user requesting the data. This option is mostly used when you are working with a lookup SAP system, or where user-level security is not required.

Behind the scenes, the Service Instance will be set to use Static authentication and the username and password will be encrypted and stored in the Service Instance.

K2 SSO

This option will utilize the K2 Single Sign-On mechanism to cache and resolve user credentials for SAP at runtime. Users will need to maintain their cached user credentials in the K2 workspace tooling and specify the SAP destination name that the credentials are used for. Alternatively, developers can write code to perform the caching. We will cover this approach in more detail in the next topic, and for more information refer to the following K2 KB article: <http://help.k2.com/en/kb000360.aspx>

Manually caching credentials for SAP destination in K2 workspace

SAP username + password

Destination Name

SAP SSO

With the SAP SSO approach, K2 will require that users pass a SAP Logon Ticket as a parameter whenever they call a SmartObject method. The Logon Tickets are obtained from a SAP system (for example a SAP NetWeaver portal).

See this K2 KB article for information on using SAP SSO: <http://help.k2.com/en/kb000689.aspx>

See this SAP article for more on using SAP SSO Logon Tickets: <http://scn.sap.com/docs/DOC-17126>

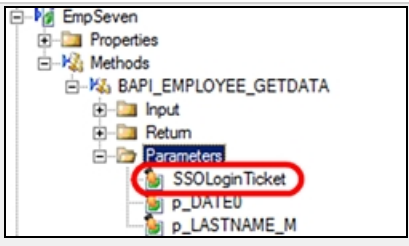
Note

When using SAP SSO, developers will need to create a mechanism to retrieve and store a user's Logon Ticket.

One approach would be to create a Service Broker which generates a logon ticket for a user and then passes this logon ticket to the SmartObject parameter.

You can then create composite SmartObjects which call the service broker first to retrieve a token for the user, and then pass this token into the K2 connect Service Object's SSOLoginTicket input parameter.

The SSOLoginTicket parameter which is added to all SmartObject method calls when using SAP SSO runtime security



Manually adding Service Instances

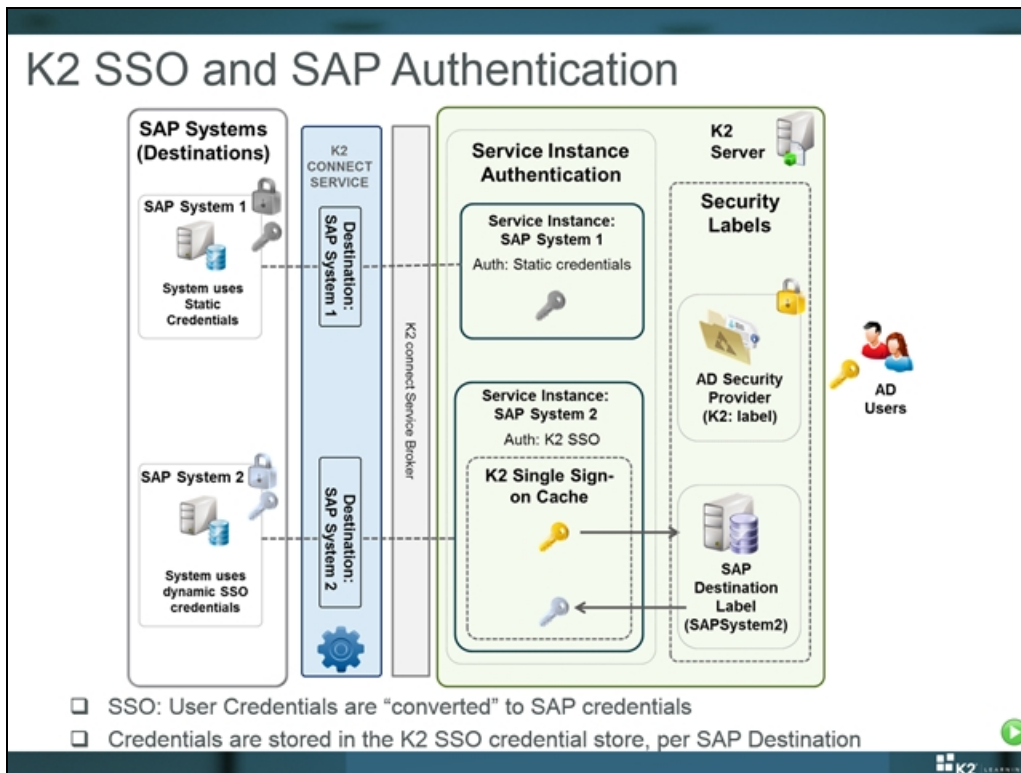
If you need to register Service Instances for K2 connect destinations manually, you will need to enter the SAP authentication mechanism when registering the Service Instance. The screenshot below illustrates the various authentication options available.

Connect Service Instance configuration

The screenshot shows the 'Add Service Instance' dialog box with the following configuration and annotations:

- Authentication Mode:** Set to 'Static'. Annotation: "Static" or "SSO" for K2 connect SAP Service Instances linked to K2 connect Destinations
- Security Provider:** (Empty dropdown)
- User Name:** k2sapdev. Annotation: If using "Static" authentication, use the SAP username and password here
- Password:** (Empty field) with a checked 'Retain Existing Password' checkbox.
- Extra:** SAP-DEV. Annotation: Name of the K2 Connect Destination
- Service Keys:**
 - server - Required: dx
 - port - Required: 8085
 - sapso - Required: false. Annotation: If using SAPSSO, this setting will be "true". This adds a input paramter for SAP Logon Ticket for all SmartObject calls
 - typemapping: (Empty field)

K2 SSO and SAP Authentication



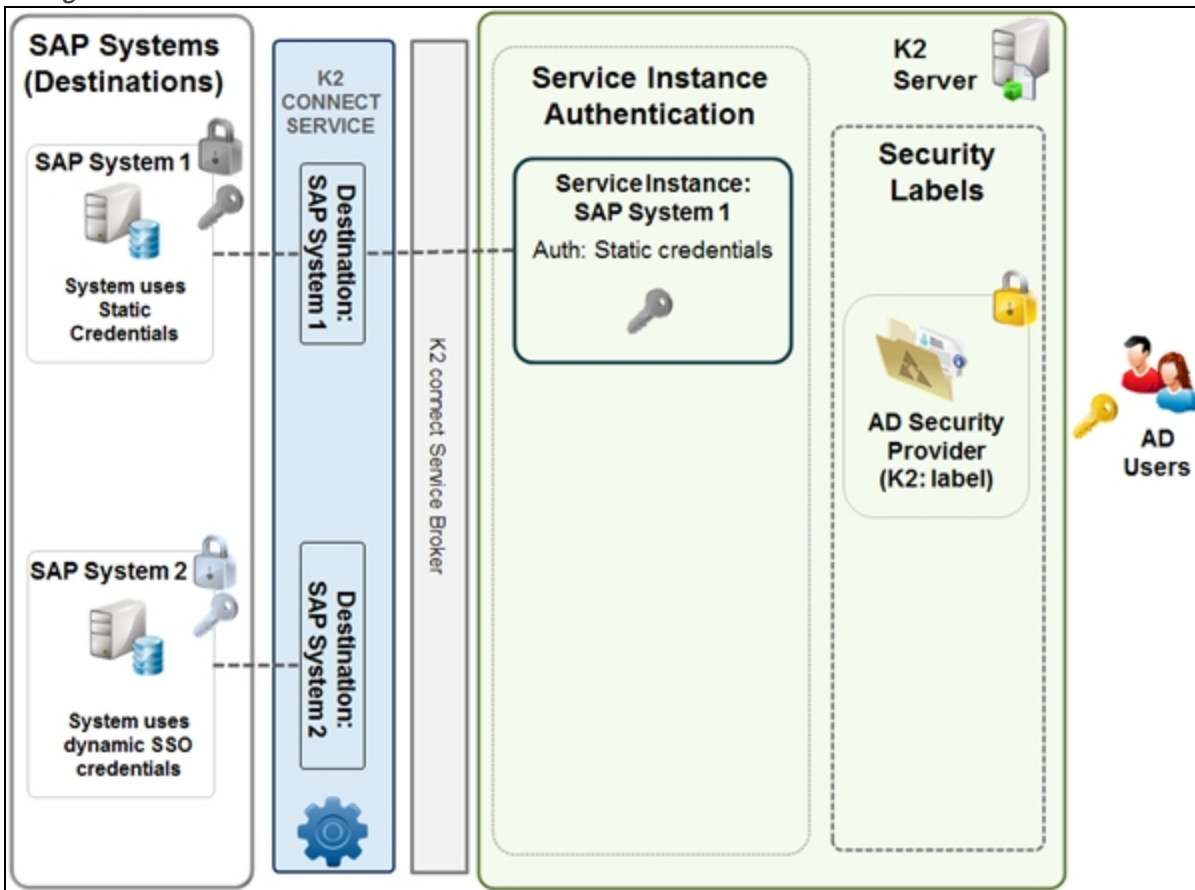
Let's look at the two most commonly-used SAP authentication mechanisms: Static credentials and K2 SSO.

Static Credentials

When you use Static credentials for a connect Destination and its associated Service Instance, users will authenticate against K2 with their normal user credentials. (In most cases, this will be their Active Directory credentials). Since the Service Instance uses Static credentials, the user credentials effectively stop at the service instance and further communication happens in the context of the static credentials.

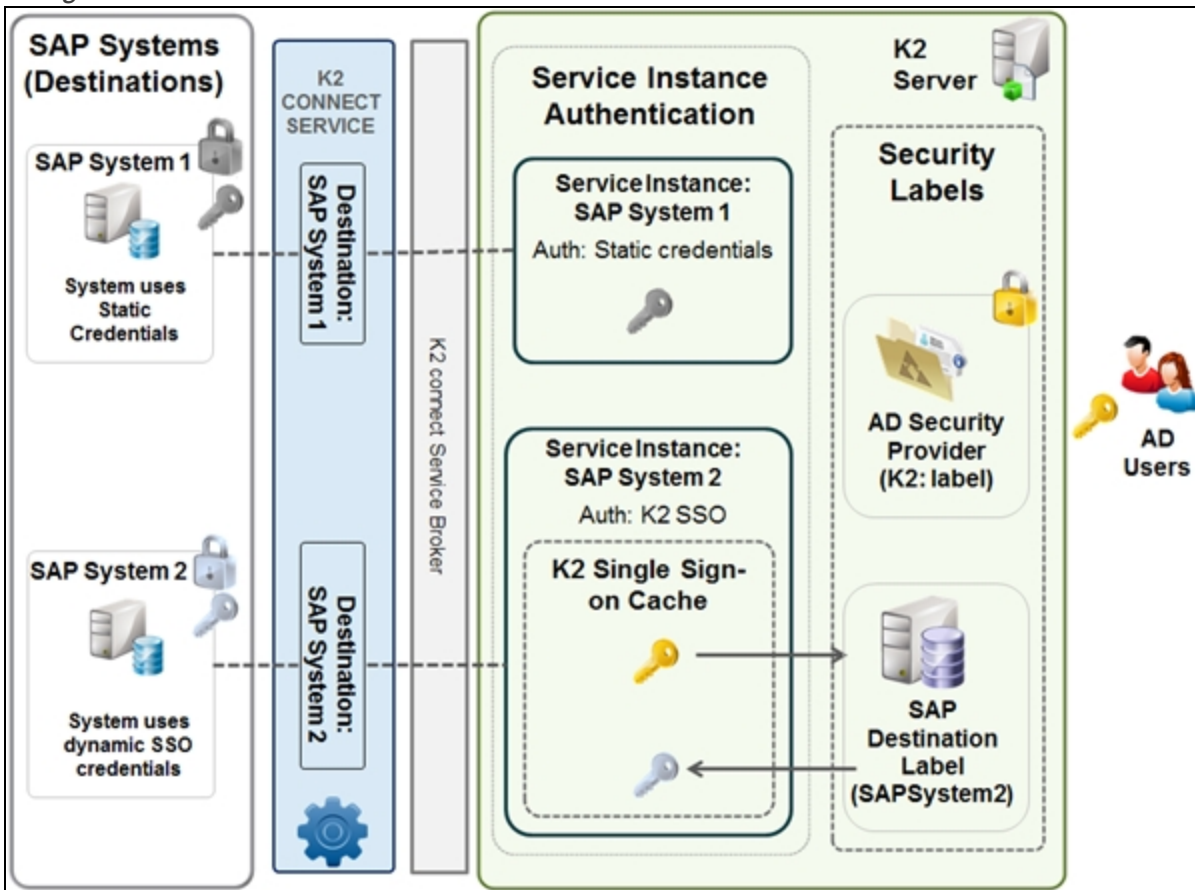
Consider the diagram below, which illustrates the static authentication approach. The user credentials (represented by the golden key) are only used with the built-in K2 security provider for Active Directory to authenticate the user. From there, the call transitions to the static credentials defined for the Service Instance (represented by the grey key) and those credentials are used to communicate with the **SAP System 1** Destination. The Destination passes these credentials through to the underlying SAP system. It should be clear that all interaction between K2 and SAP occur in the security context of the static credentials, and the SAP system essentially has no context of the user that connected to K2.

Using Static credentials for a service instance and Destination



K2 SSO

When using K2's Single Sign-On mechanism, user credentials are automatically and transparently translated into SAP credentials by K2. Consider the diagram below, which illustrates this concept.



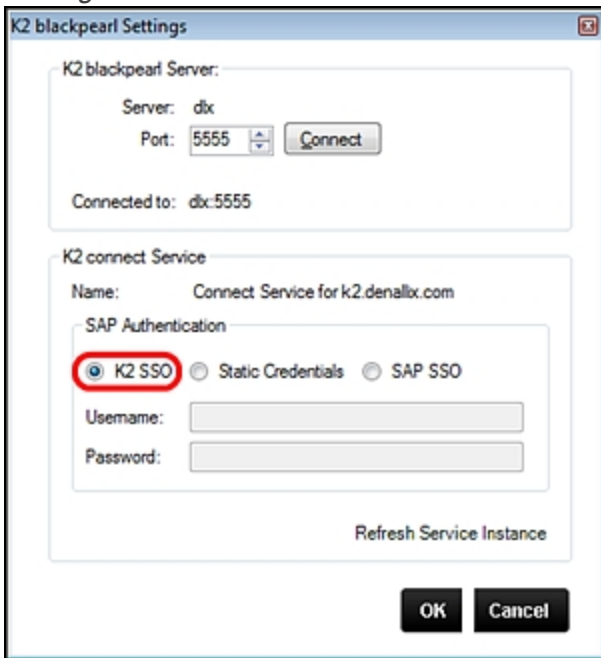
The second Service Instance (**SAP System 2**) and its associated K2 connect Destination uses K2 SSO as the authentication mechanism. As before, users authenticate with K2 using their Active Directory credentials (represented by the golden-colored key). When these users attempt to execute a method for a SmartObject tied to the second Service Instance, K2 notices that the Service Instance uses Single Sign-On, so it interrogates the single sign-on store for the target SAP Destination to locate a set of cached credentials for the current user. If no cached credentials are found for the Destination, an error is returned to the user. K2 uses the Destination Name to locate a Security Provider where the **Security Provider Label** and **Extra Data** settings match the name of the Destination.

If credentials are located, K2 will retrieve the credentials for the user (represented in this diagram with the blue-colored key). These SAP credentials are then passed through the K2 connect destination and the user is authenticated on the SAP system with their own SAP credentials. This approach means that SAP does have context of the user that originally connected to the K2 server, and any SAP security trimming is maintained.

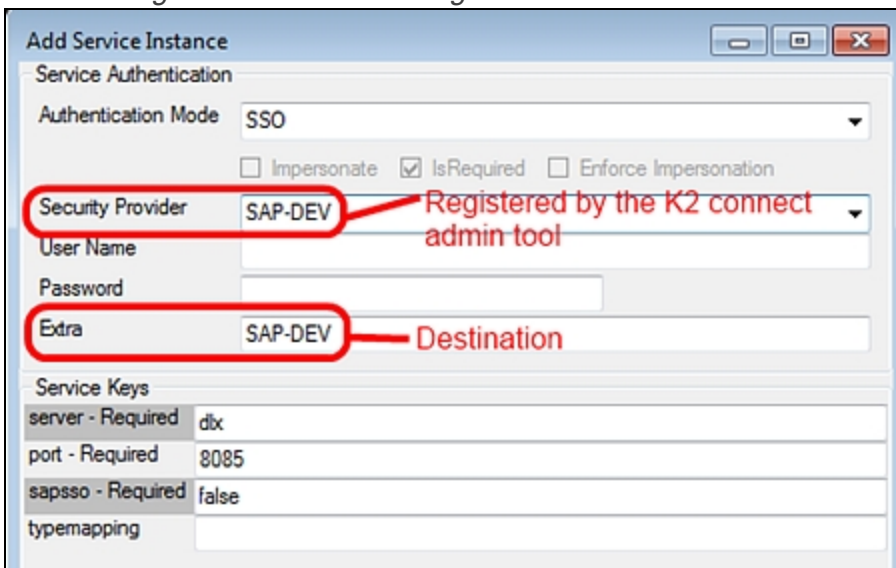
The following screenshots illustrate how K2 SSO is configured with K2 connect destinations. The recommended approach is to use the K2 connect administration tool and select K2 SSO as the SAP Authentication mechanism.

When you select **K2SSO**, the administration tool will register a new Security Provider in the K2 system with the same name as the Destination. The Service Instance is then automatically associated with the new Security Provider entry.

Setting K2 SSO as the authentication mechanism

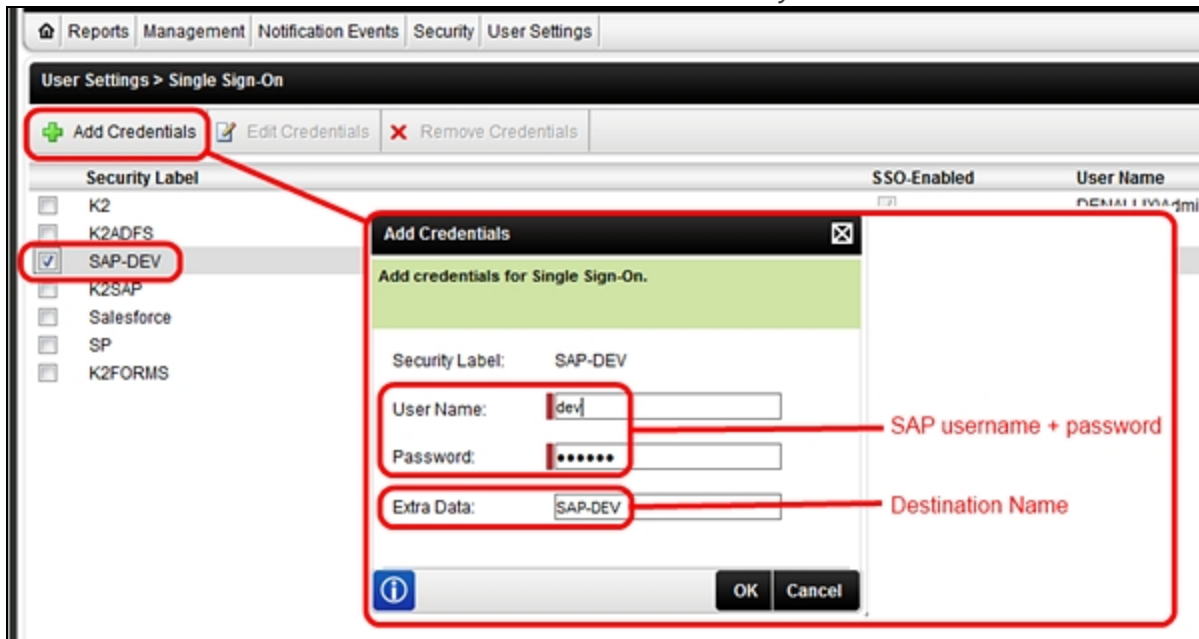


The resulting Service Instance configuration



Once the Service Instance is configured, users will configure their cached credentials using K2 workspace, as shown below. The user needs to select the security provider for the target SAP system, and then **Add Credentials** for the provider. On the Add Credentials screen, it is important that the user enters the K2 connect Destination Name in the **Extra Data** field, as shown in the screenshot below.

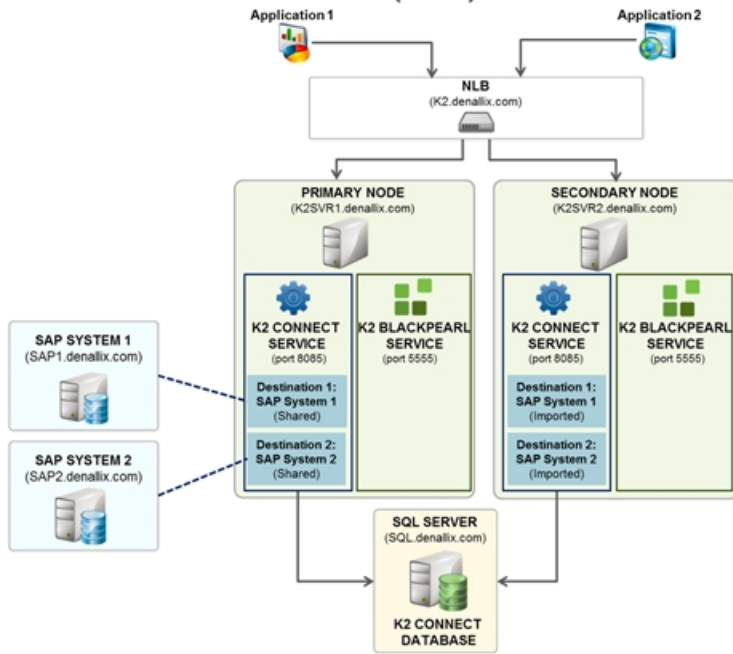
User caches credentials for K2 SSO for the SAP-DEV Security Provider



Once the credentials are cached, they allow the user to interact with the SmartObject as they normally would, and K2 will handle the underlying credential conversion transparently.

Distributed Install - NLB

Distributed Install - NLB (1/2)



Distributed Install - NLB (2/2)

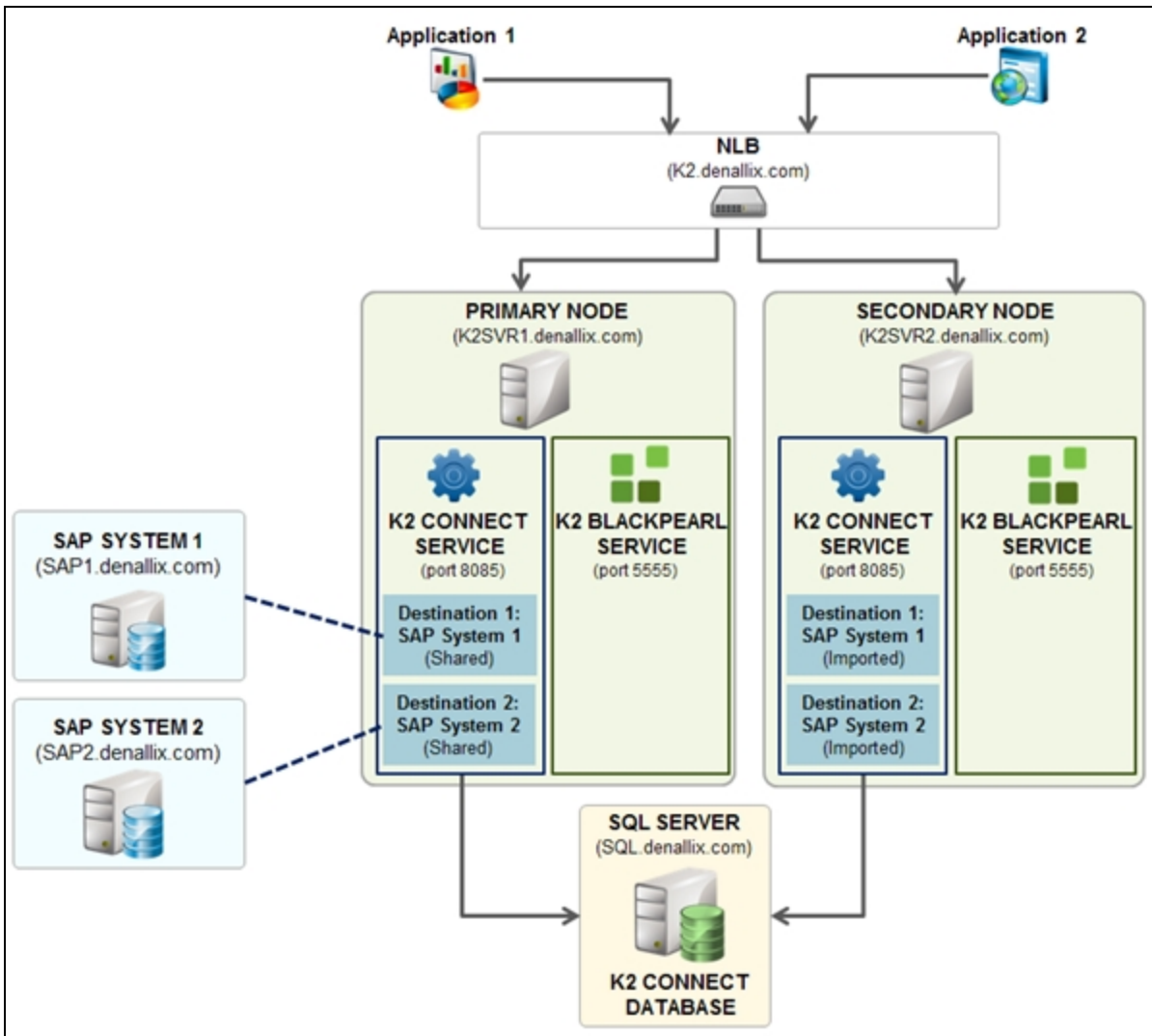
- Current version requires some manual configuration for NLB setups
- See this KB for details: <http://help.k2.com/en/kb000410.aspx>
- Installation Overview:
 1. Install connect database only from a non-K2 machine
 2. Install *librfc32.dll* and *ERP connect* on each K2 server in the cluster
 3. Install K2 connect server on 1 K2 server in the cluster ("primary node")
 4. Set the connect License key for the primary node
 5. Set up a shared Destination on the primary connect server
 6. Install connect server on each other K2 server in the cluster ("secondary node")
 7. Set License key separately for each secondary node
 8. Import the shared Destination from the primary node and set it as default

It is possible to install K2 connect in a distributed, NLB environment. We will briefly describe the approach and some considerations when installing K2 connect in a NLB environment.

NLB architecture

A distributed K2 connect installation looks very similar to a distributed K2 blackpearl installation. Consider the diagram below, which describes a hypothetical NLB environment.

K2 connect in a NLB environment



In this environment, we have two physical K2 servers (**K2SVR1** and **K2SVR2**) which are exposed to other applications as a logical machine (**K2**) using Network Load Balancing (NLB). The NLB system will distribute requests addressed to **K2.denallix.com** between the two physical K2 servers.

Each K2 server has an installation of K2 connect and K2 blackpearl. Notice that the services run on different ports (5555 for K2 blackpearl and 8085 for K2 connect; these ports can be changed if needed). The K2 connect installations share the same database on the **SQL.denallix.com** SQL server. We also have two SAP systems (**SAP1** and **SAP2**) which are configured as destinations on the K2 connect environment. Each of our K2 connect servers knows about each of the SAP systems.

The main thing to understand with an NLB installation is how the installation works and how destinations are shared between the K2 connect installations.

Distributed installation considerations

At the time of writing, a distributed installation of K2 connect requires some manual steps. The KB article <http://help.k2.com/en/kb000410.aspx> describes the steps in more detail, so we will only describe the high-level steps here. These steps assume that you already have an operational NLB setup and that K2 blackpearl has been installed and configured.

1. Install connect database only from a non-K2 machine.

This action will set up the K2 connect database on a specific SQL server without installing any other K2 connect components. We recommend performing this action on a machine other than the K2 servers, because you will run the server installation separately on each of the K2 servers.

2. Install ERPConnect and librfc32.dll on each K2 server in the cluster.

As for normal installation, install the prerequisites for K2 connect on each server.

3. Install K2 connect server on 1 K2 server in the cluster ("primary node").

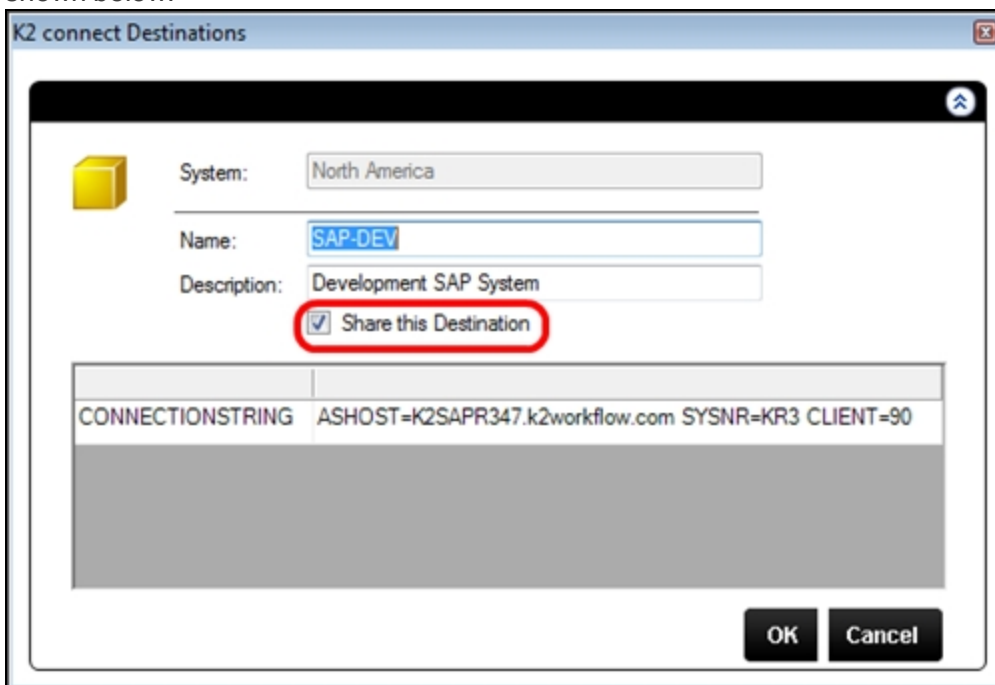
Install the K2 connect server components on one machine in the NLB farm. (We'll call this machine the "Primary node") and point it to the existing connect database that was installed in step 1.

4. Set the connect License key for the primary node.

K2 connect servers are licensed individually, so obtain and set the license key for the primary node. Note that the license keys are hardware-linked, so you cannot use the same key for all your connect servers.

5. Set up a shared Destination on the primary connect server.

Configure the Destinations on the primary node. Importantly, you should mark the destination as "shared", as shown below:



Test the destination to make sure that the connection string is correct.

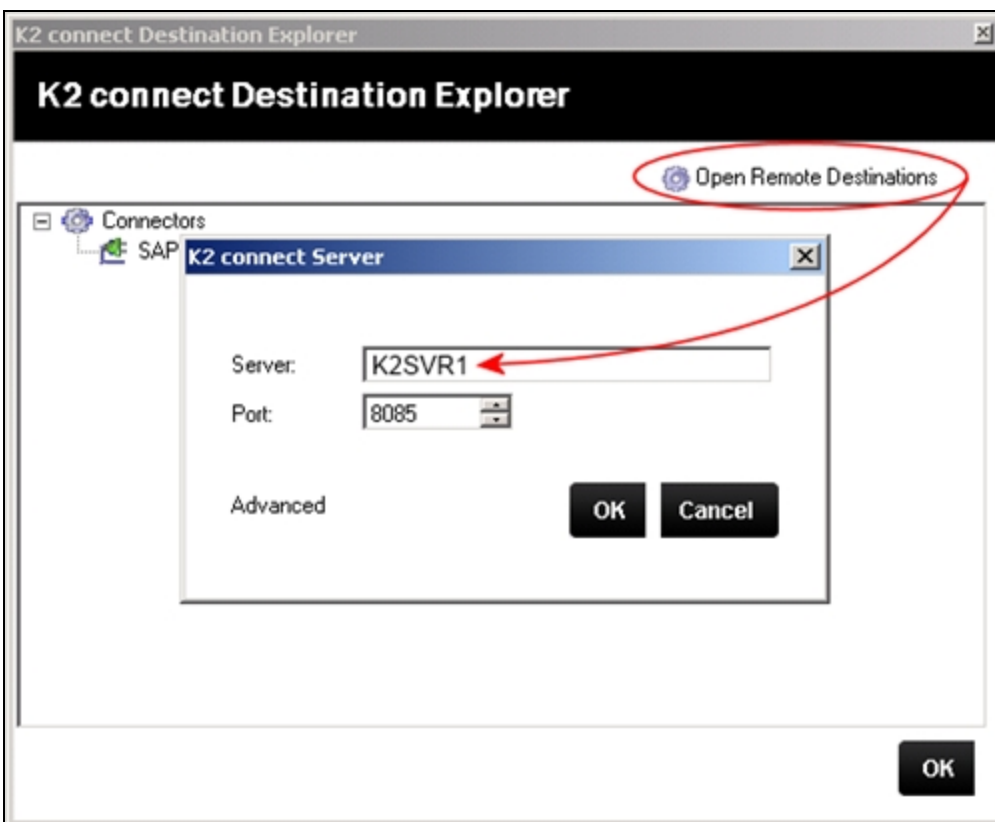
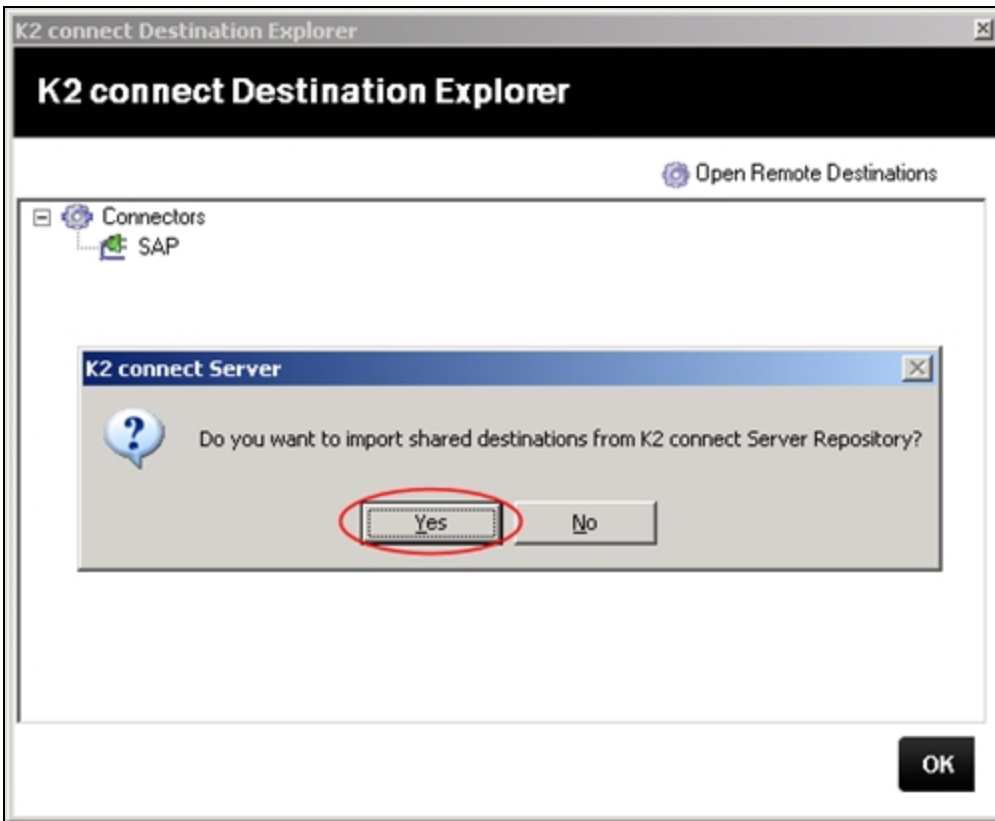
6. Install connect server on each other K2 server in the cluster ("secondary node")

Install the K2 connect server components on each of the other nodes in the cluster (we'll call these the "secondary nodes" and point them to the same K2 connect database that was installed in Step 1.

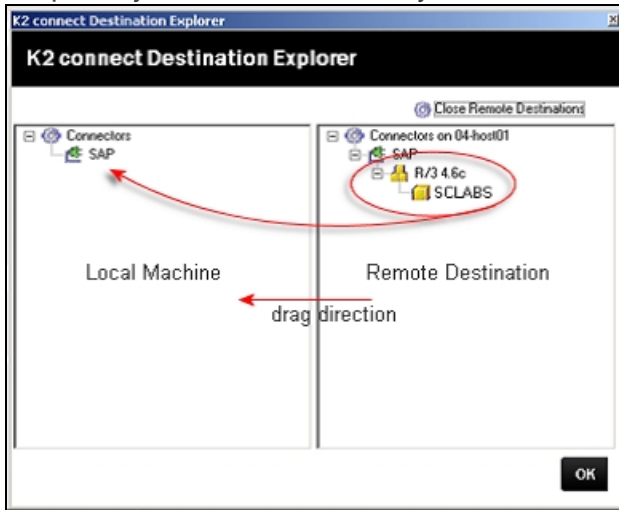
7. Set the connect License key separately for each secondary node

8. For each secondary node, import the shared Destination from the primary node and set it as default

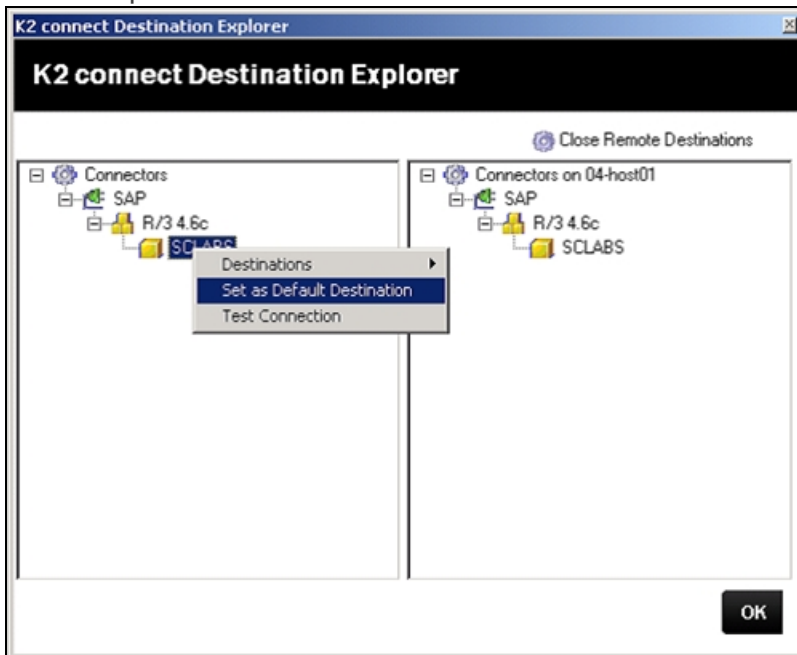
You will be prompted to import shared Destinations from a repository. Select Yes and then use the Open Remote Destination button to connect to the primary node (you will need to use the physical machine name to connect).



9. The Destination explorer will launch, and you can then drag and drop the shared Destinations from the primary node to the secondary node.



10. Set the imported destination as the Default destination on each secondary node.



Logging and Troubleshooting

Trouble Area	Troubleshooting Tools	Logging
SmartObjects and Service Objects layer	<ul style="list-style-type: none"> ▪ K2 SmartObject Tester 	<ul style="list-style-type: none"> ▪ SmartObject Logging ▪ Host Server Console mode
K2 connect layer	<ul style="list-style-type: none"> ▪ K2 connect Administration Tools (connectivity/connection string) ▪ K2 connect Test Cockpit ▪ ERP Connect Test Code (see documentation for samples) 	<ul style="list-style-type: none"> ▪ Host Server Console mode ▪ Protocolling on K2 connect server (enable protocolling) ▪ Windows Event viewer ▪ Return codes in Test Cockpit ▪ RFC Exception trace ([Program Files]\K2 connect\admin\dev_rfc.trc)
SAP BAPI layer	<ul style="list-style-type: none"> ▪ SAP GUI ▪ ABAP Debugging (if SAP GUI is installed) 	<ul style="list-style-type: none"> ▪ SAP Logging ▪ ABAP Debugging (if SAP GUI is installed)

• SAP Errors are returned either as return codes or exceptions

Between applications, K2 SmartObjects and the K2 connect architecture, troubleshooting issues with SAP BAPIs exposed as K2 SmartObjects can get complex, especially when you consider that trouble can occur at many different levels: on the SmartObjects layer, on the K2 connect layer or in SAP itself.

The best way to troubleshoot issues with SAP-backed SmartObjects is to use the available tools and logging mechanisms to isolate the cause of the problem. The table below lists some of the common tools used to troubleshoot issues and the logging mechanisms available, based on the area where the trouble may exist.

Note

Note that errors in SAP could be returned as RETURN codes, in which case the SmartObject may not report an error, or as SmartObject exceptions, in which case the SAP error will be bubbled up all the way to the consuming application.

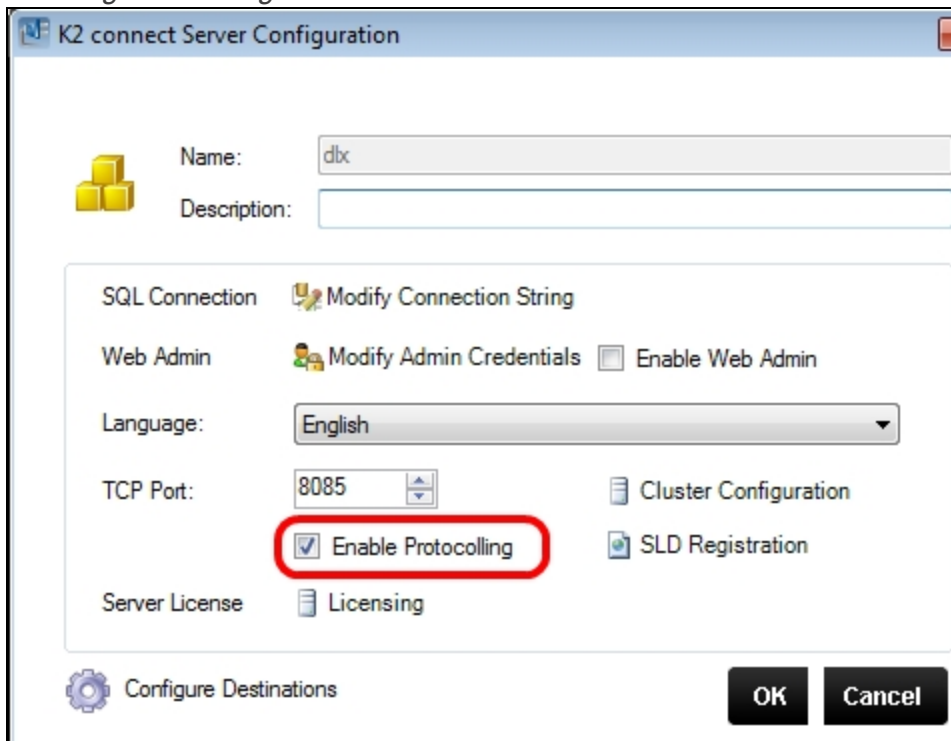
Trouble Area	Troubleshooting Tools	Logging
SmartObjects and Service Objects layer	<ul style="list-style-type: none"> ▪ K2 SmartObject Tester [Program Files]\K2 blackpear\Bin\SmartObject Service Tester.exe 	<ul style="list-style-type: none"> ▪ SmartObject Logging enable logging in the file [Program Files]\K2 blackpear\Host Server\Bin\SourceCode.SmartObjects.Runtime.config. Log files are output by default to [Program Files]\K2 blackpear\ServiceBroker\logs ▪ Host Server Console mode run K2 server in console mode in the context of the K2 service account, and review console output.
K2 connect layer	<ul style="list-style-type: none"> ▪ K2 connect Administration Tools Use the K2 connect administration console to Test the connection to the target SAP system ▪ K2 connect Test Cockpit 	<ul style="list-style-type: none"> ▪ Host Server Console mode/Log files Configure host server logging (file [Program Files]\K2 blackpear\Host Server\Bin\HostServerLogging.config) and review the log output for errors. ▪ Protocolling on K2 connect server Enable protocolling on the K2 connect server, and

	<p>Use the test cockpit and execute the BAPI. Check the RETURN value to see if any errors were reported</p> <ul style="list-style-type: none"> ■ ERPConnect Test Code If needed, write code using the ERPConnect adapter, which will exclude K2 connect from the equation. See the connect product documentation and ERP documentation for code samples 	<p>review the output. (See section below for details on enabling protocolling)</p> <ul style="list-style-type: none"> ■ Windows Event viewer The windows event log may contain useful information. Remember to check the Application and Security logs. ■ Return codes in Test Cockpit The RETURN property of a BAPI in the Test Cockpit can contain an error message from SAP which may help you locate the source of the problem. ■ RFC Exception trace ERPConnect will output a trace file with errors. This file can be located at <i>[Program Files]\K2 connect\admin\dev_rfc.trc</i> or <i>[Program Files]\K2 blackpearl\Host Server\Bin\dev_rfc.trc</i>. Open the file with a text editor to see if any useful error information was written.
SAP BAPI layer	<ul style="list-style-type: none"> ■ SAP GUI If available, use the SAP GUI to execute the BAPI to determine whether the problem is on the BAPI layer. ■ ABAP Debugging (if SAP GUI is installed) You can also use ABAP debugging to troubleshoot BAPI-level errors. 	<ul style="list-style-type: none"> ■ SAP Logging Ask your SAP administrator to review any SAP logfiles that may have useful troubleshooting information

Enabling Protocolling for a Connect Server

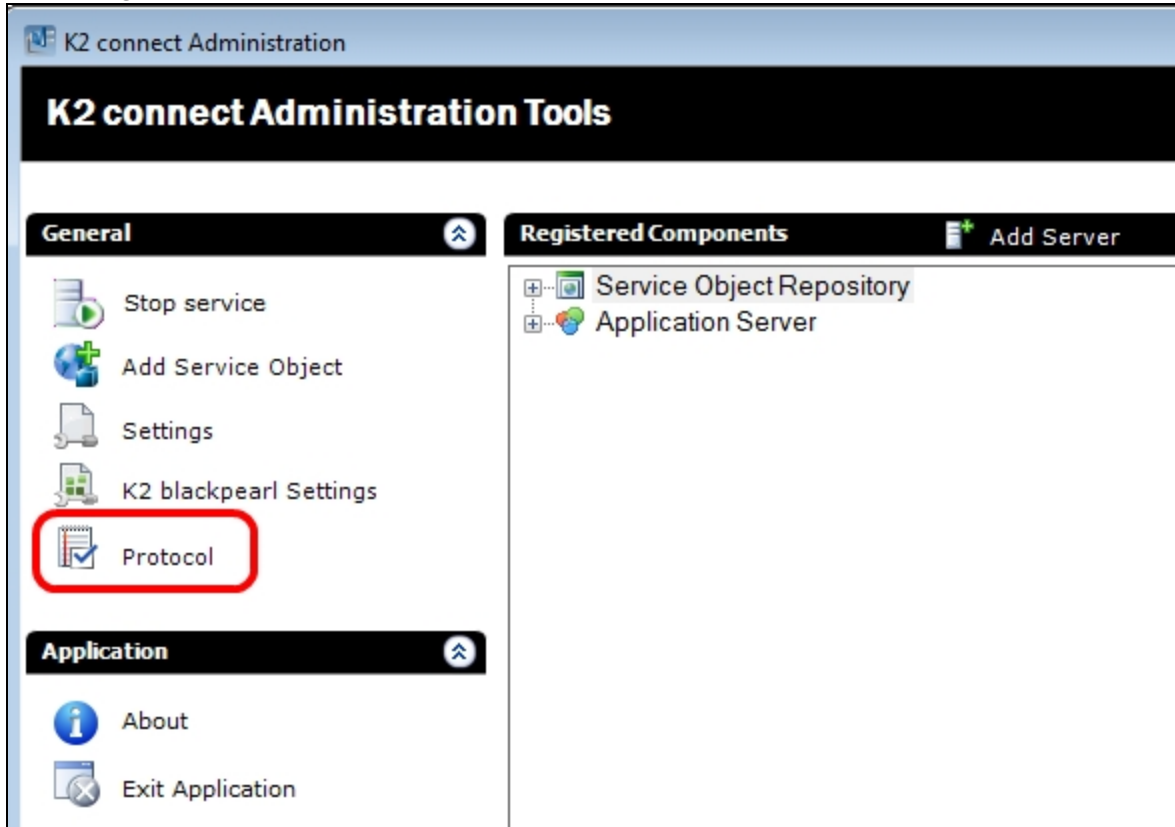
Protocolling is essentially a real-time output of the BAPIs that are called by K2 connect. To enable protocolling output, open the K2 connect Administration console's Settings window. Select the Enable Protocolling checkbox and restart the K2 connect service.

Enabling Protocolling

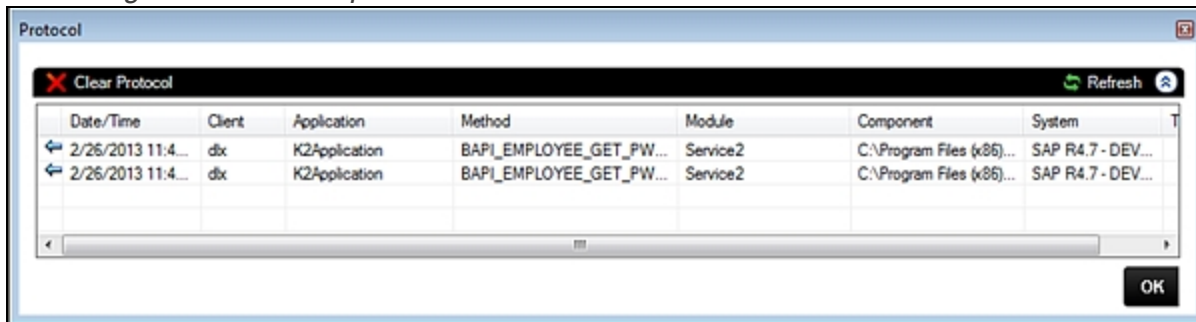


Once you have restarted the K2 connect service, use the SmartObject Service Tester utility to execute the SAP-backed SmartObjects, and then open the Protocol window of the K2 connect administration console. Review the output of the Protocol window and see if any useful troubleshooting information was logged.

Accessing the Protocol window



Reviewing the Protocol Output



Review and Q&A

Review and Q&A

- K2 connect and SAP conceptual architecture
- Installing K2 connect and prerequisites
- SAP systems, connect Destinations and K2 Service Instances
- Configuring Destinations
- Multiple Destinations and shared SAP environments
- Design-time authentication vs. Runtime authentication
- Static credentials vs. SSO credentials
- Testing SAP BAPIs
- Service Object Explorer
- Test Cockpit
- Logging and troubleshooting

Let's review the main topics covered in this learning module:

- K2 connect and SAP conceptual architecture
- Installing K2 connect and prerequisites
- SAP systems, connect Destinations and K2 Service Instances
- Configuring Destinations
- Multiple Destinations and shared SAP environments
- Design-time authentication vs. Runtime authentication
- Static credentials vs. SSO credentials
- Testing SAP BAPIs
 - Service Object Explorer
 - Test Cockpit
- Logging and troubleshooting

Think about your own SAP systems and the BAPIs you may want to expose as SmartObjects. Can you identify any potential composite SmartObjects that would combine data from SAP and other systems? How many SAP systems would you want to expose to K2 connect, and how would you configure the security? Would you use static credentials or one of the SSO approaches, and why? Do you understand the concept of K2 connect Destinations?

If you are in a classroom environment, use this opportunity ask the instructor to explain or elaborate on any topic that is unclear. If you are in a self-study situation, check out the Additional Resources listed below. The next module will cover K2 connect development in more detail, so your question may be deferred until you have completed the next module.

Additional Resources

The following table lists additional resources that supplement the information provided in this module:

Resource	Location and Notes
K2 Connect in a NLB environment	http://help.k2.com/en/KB000410.aspx
Installing lib32rfc.dll in a 64-bit environment	http://my.theobald-software.com/index.php?/Knowledgebase/Article/View/71/0/theobald-products-in-a-64-bit-environment
K2 connect quick install guide	http://help.k2.com/en/KB000707.aspx

<p>How to cache credentials for K2 connect for SAP Security Labels</p>	<p>http://help.k2.com/en/kb000360.aspx <i>Describes how to use K2's SSO feature to cache runtime user credentials for SAP. Both the manual approach and code-based approaches are covered.</i></p>
<p>SAP SSO Support</p>	<p>http://en.wikipedia.org/wiki/SAP_Logon_Ticket <i>Overview of SAP Single Sign-On</i> http://help.k2.com/en/KB000689.aspx <i>Describes how to enable and use SAP SSO based on SAP Logon Tickets.</i> http://scn.sap.com/docs/DOC-17126 <i>Describes how to use and set up SAP SSO Logon Tickets for Microsoft-based systems</i></p>
<p>ERPConnect FAQ's</p>	<p>http://my.theobald-soft-ware.com/index.php?/Knowledgebase/Article/View/55/3/erpconnect-faqs <i>Some common questions and answers about ERPConnect</i></p>

Developing with K2 connect



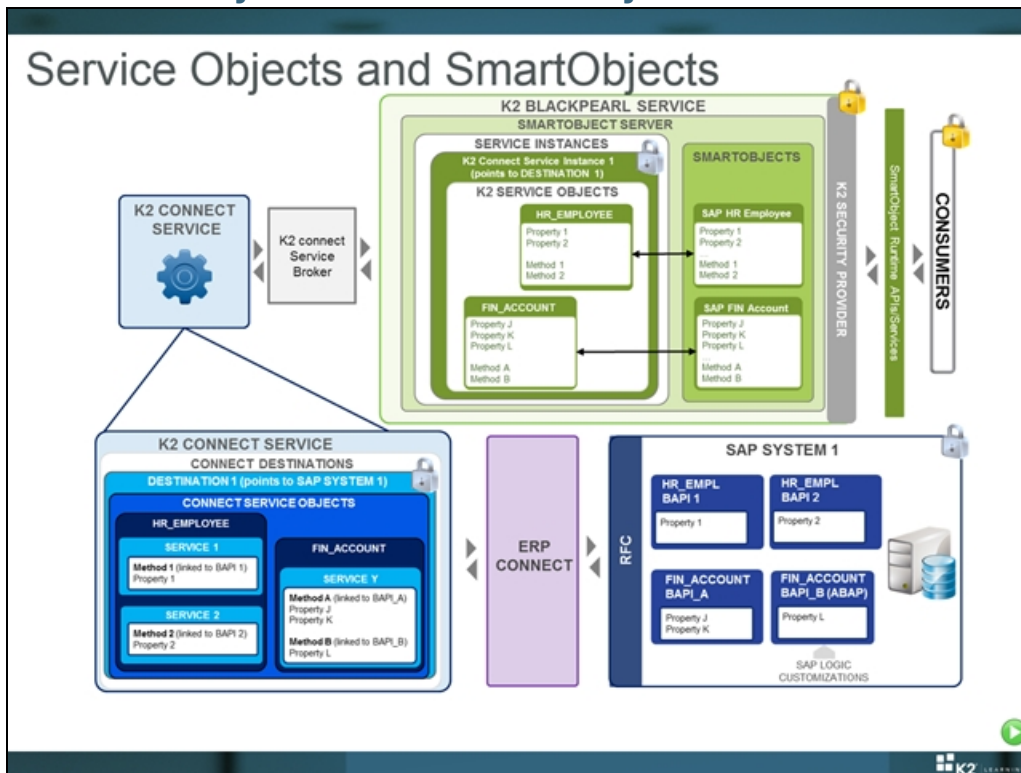
This learning module describes how developers use the various K2 connect development tools to create connect Service Objects and K2 SmartObjects so that SAP BAPIs can be exposed to consuming applications. This module is intended for all roles that will develop with K2 connect. Typical roles include K2 Developers, Solution architects, and SAP Developers.

At the end of this module, participants should have a solid understanding of how to develop K2 connect Service Objects and K2 SmartObjects to expose SAP BAPIs. Participants should also understand some of the more advanced design and deployment considerations that are encountered in real-world K2 connect implementations.

Tip

A Recorded video of this module is available at <https://youtu.be/-ODXej989EQ>

Service Objects and SmartObjects



Before you start developing with K2 connect, it is important to understand the basic principles and terms used to describe the components that form the integration stack between SmartObjects and SAP. We discussed the architecture in prior connect learning modules, but let's look a little deeper into the relationship between BAPIs, connect Service Objects, K2 Service Objects, and SmartObjects.

Consider the diagram below, which illustrates these concepts and terms.

In the bottom right corner we have an SAP system with several BAPIs which we would ultimately like to expose as K2 SmartObjects. These BAPIs could be standard SAP BAPIs or could be custom ABAP BAPIs. (Remember that BAPIs need to be RFC-enabled before you can use them in K2 connect).

If you think back to the previous module, you should recall that K2 connect is configured with Destinations, which allows connect to interact with a particular SAP system. In our diagram, we have a Destination (**DESTINATION 1**) which has been configured to point to a SAP environment (**SAP SYSTEM 1**).

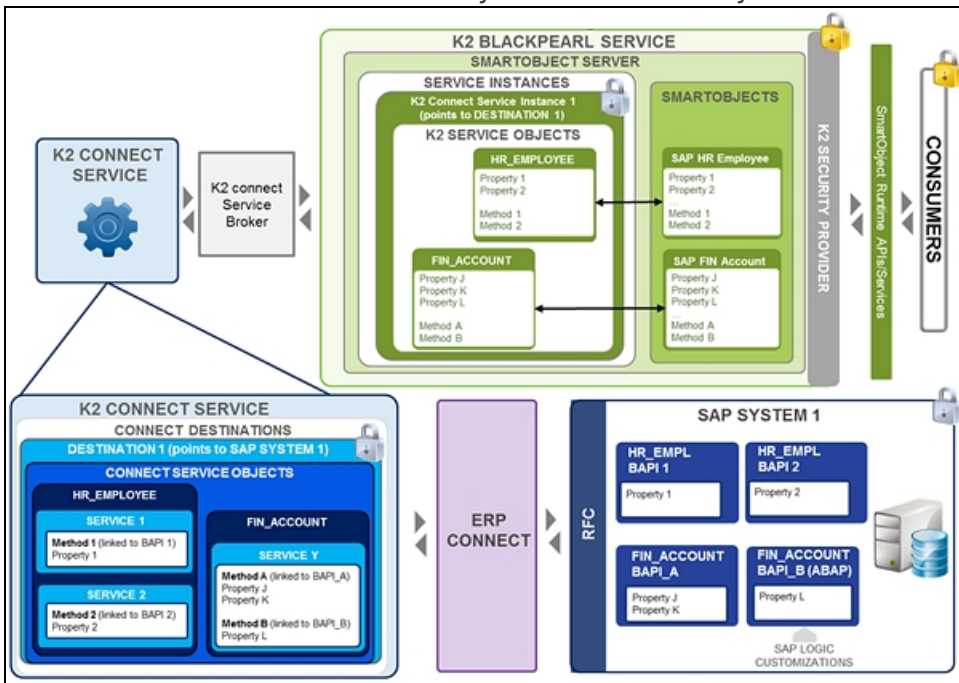
At this point, a developer can use the connect development tools in Visual Studio to create K2 connect Service Objects for the configured Destination. During this process, the developer would create connect Service Objects by dragging and dropping selected SAP BAPIs into their Service Object or by defining BAPI mappings manually. They can combine multiple BAPIs into one logical Service Object, or they may decide to separate the Service Object's methods into separate Services. Don't worry too much about the details just yet: for now, it's only important to understand that connect Service Objects contain one or more Service Methods, which point to SAP BAPIs.

Consider the connect Service Objects in the diagram. Here, a developer created two connect Service Objects (**HR_EMPLOYEE** and **FIN_ACCOUNT**). The **HR_EMPLOYEE** connect Service Object has two Services (**SERVICE 1** and **SERVICE 2**), and each Service contains a single Method (**Method 1** and **Method 2**, respectively). The Service methods have been linked to equivalent SAP BAPIs - **Method 1** points to the BAPI **HR_EMPL BAPI 1** and **Method 2** points to **HR_EMPL BAPI 2**.

The **FIN_ACCOUNT** connect Service Object has a single service (**SERVICE Y**) and this service contains two methods (**Method A**, which is linked to the **FIN_ACCOUNT BAPI_A** and **Method B**, which is linked to the BAPI **FIN_ACCOUNT_BAPI_B**).

(descriptive text continues after the diagram...)

K2 connect architecture with SmartObjects and Service Objects



Once the developer has designed the connect Service Objects with Visual Studio, he or she would publish these Service Objects to a K2 connect server.

If you think back to the previous learning module, you will recall that K2 connect Destinations have an associated K2 Service Instance. This is an important point, because the Service Instance is used to expose the connect Service Objects to K2 design tools so that designers can create K2 SmartObjects. Since the Service Instance points to a specific connect Destination, it follows that the Service Instance essentially connects to a specific SAP system. Look at the diagram again: here, you should be able to see that the **K2 connect Service Instance 1** under **Service Instances** is configured to point to the K2 connect **DESTINATION 1**. The little silver padlock icon indicates that the Service Instance (and the associated Destination) has been set up with a specific authentication mechanism (static or SSO, as we discussed in the previous learning module).

When a developer publishes his or her new or updated connect Service Objects to the K2 connect server, the Service Instance for the Service Objects' Destination is usually refreshed. During this refresh process, the K2 SmartObject server uses the Service Instance to discover the connect Service Objects for the targeted Destination, and generates new or updates the K2 Service Objects for the Destination. (This discovery and generation process is usually transparent and handled automatically by the K2 server, but administrators or developers can manually refresh the Service Instance as well.)

In the diagram, notice that there are two K2 Service Objects (**HR_EMPLOYEE** and **FIN_ACCOUNT**) which contain Properties and Methods. These properties and methods map to the K2 connect Service Object's Methods and Properties. The Methods in each Service Object point to the Methods in each connect Service Object's Services, and the properties point to the fields/properties that are returned by each Service.

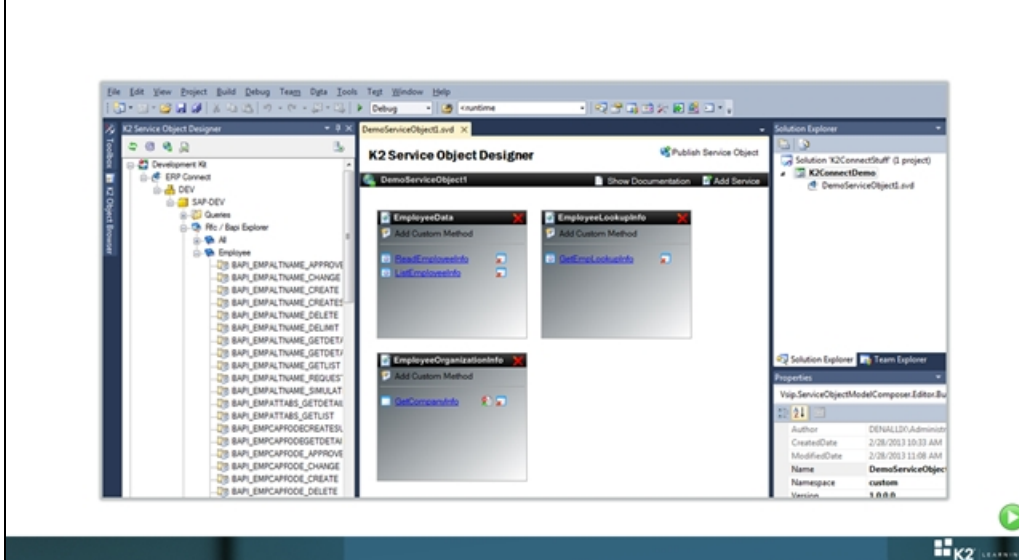
Now that the K2 Service Objects are available in the K2 SmartObject server, K2 developers or designers can create K2 SmartObjects for those Service Objects. Once these SmartObjects are published to a K2 server, they can be consumed by various applications just like any other SmartObject. Moreover, just like any other SmartObject, the consumer is completely unaware that the data and interaction is actually happening on a back-end SAP system.

Notice that there are gold-color padlocks on the Consumers and the K2 blackpearl server. This means that these layers have their own authentication mechanisms, separate from the authentication mechanism used by the K2 connect Service Instance. If you think back to the previous learning module, you should remember that you could use static or SSO credentials for a connect Service Instance, but that the user's native AD credentials are not passed back to the SAP system.

K2 connect development tools

K2 connect development tools

- Service Object Designer



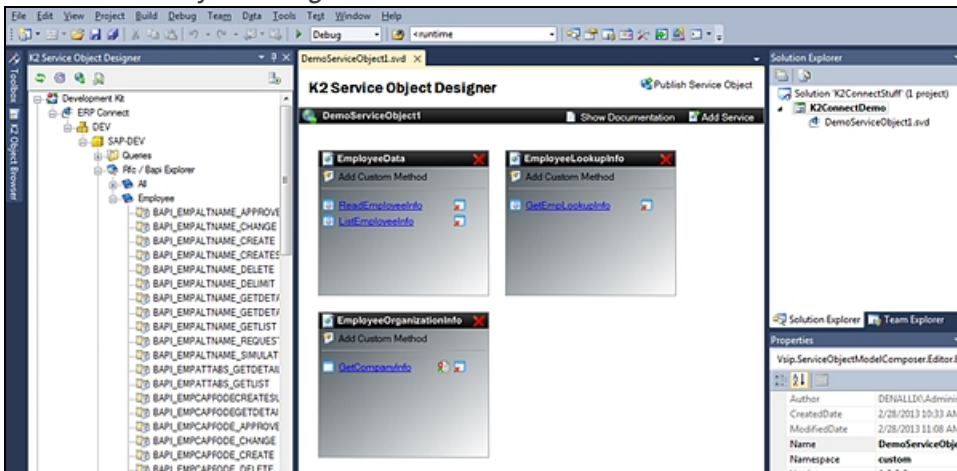
As a K2 connect developer, you will probably use a selection of tools to build and test SmartObjects based on SAP BAPIs. Let's briefly look at the most commonly used development tools:

Service Object Designer

You will use the Service Object Designer tool in Visual Studio to create new connect Service Objects. The first step is to connect to a K2 connect server, after which you can explore the available Destinations that have been defined for that server, along with the available SAP BAPIs that are available in that Destination.

When creating Service Objects, you will use a visual drag-and-drop interface to select the BAPIs that you want to expose in your Service Object. You can customize the Service Methods as well, defining custom functions, mapping properties and input parameters, write custom code to transform SAP data and more. Finally, you can use this tool to publish the Service Object to the K2 connect server. Service Objects are defined as .svd files, and must be published to a K2 connect server before you can use them; it is not possible to run a Service Object locally in debug mode.

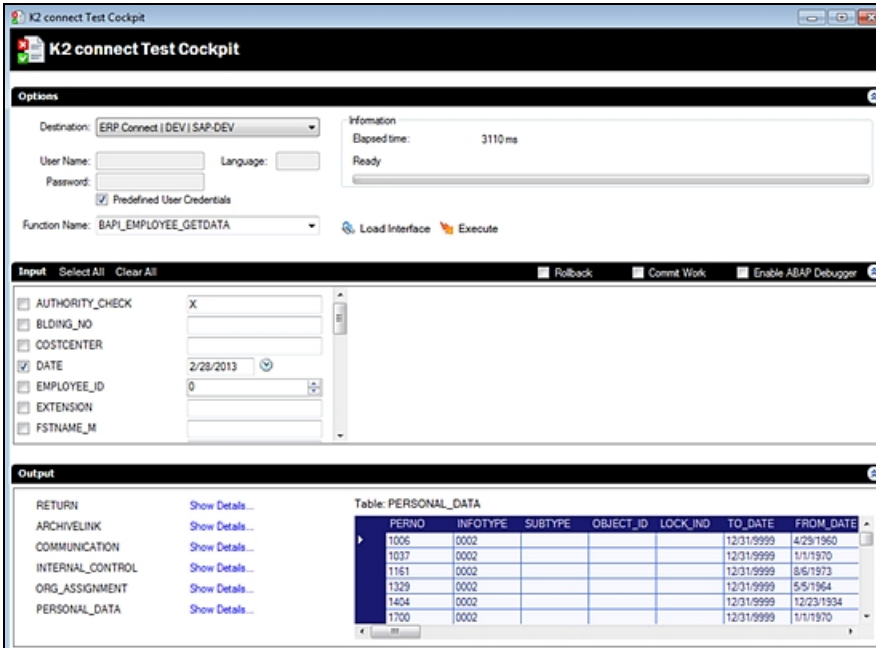
The Service Object Designer in Visual Studio



Test Cockpit

The Test Cockpit is primarily used to test and investigate SAP BAPIs. This is a nice tool to use to understand what the BAPI does, what input values it expects and what return values are available from the BAPI.

The Test Cockpit in Visual Studio



K2 connect Administration Console

Developers could use the Administration Console to configure Destinations and refresh the Service Instance for a Destination. They may restart the connect Service to pick up updated Service Object definitions, enable protocol output for debugging, and manually import Service Object Definitions (.svd's).

K2 SmartObject Service Tester utility

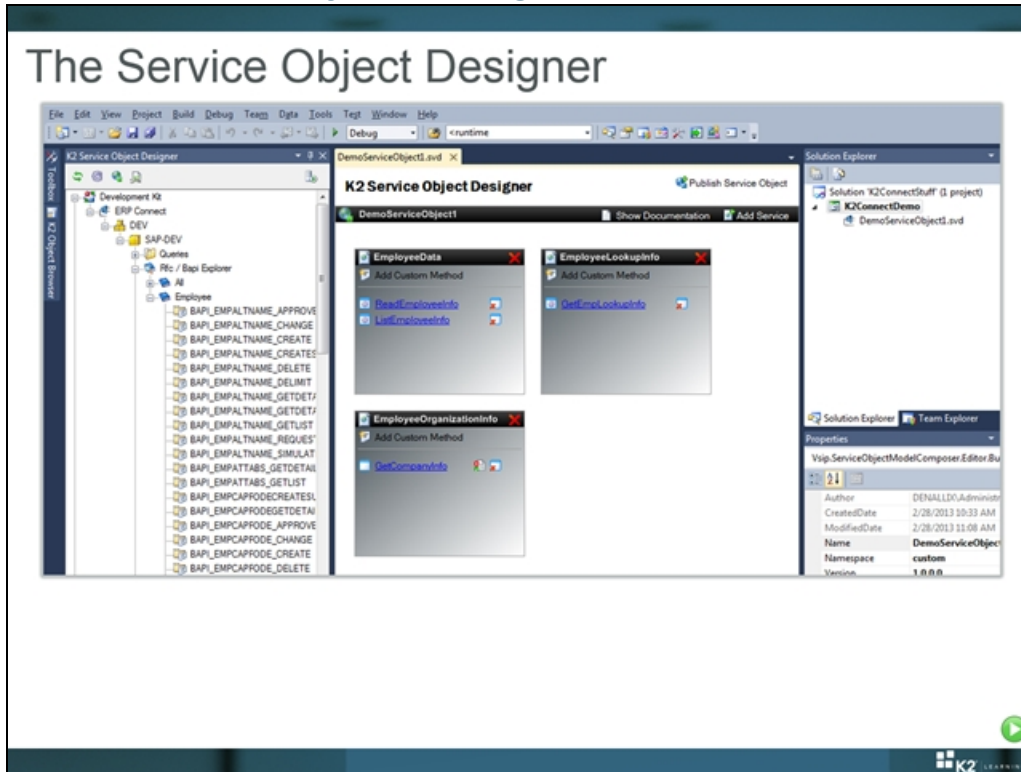
The SmartObject Service Tester utility is a very useful tool for developers since it allows you to easily generate and test SmartObjects. You will probably use this tool extensively to explore the generated Service Objects and to generate SmartObjects for testing purposes.

Note

We do not recommend that you use the SmartObject testing tool to generate SmartObjects for production, since it does not give you full control over the naming and exposed properties/methods in the SmartObject.

For SmartObjects that will be used in real solutions, the recommended approach is to create SmartObjects with K2 Studio or Visual Studio. That way, you can choose which properties and methods should be exposed on the SmartObject, you can create advanced SmartObjects, and you have full control over the SmartObject naming and the structure of the project.



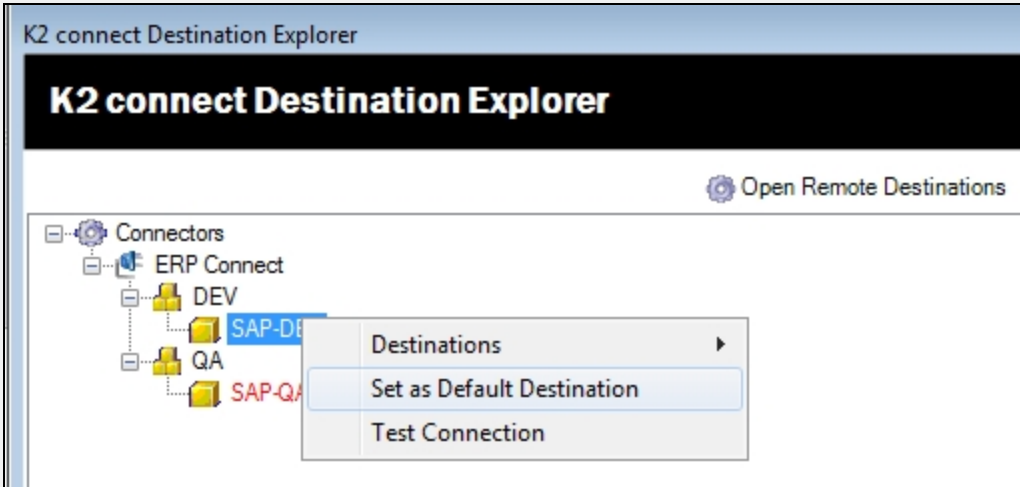

The Service Object Designer




You will be using the Service Object Designer extensively when creating SAP-backed SmartObjects, so let's have a look at the components of the designer.

Toolbar

The icons in the toolbar perform the following functions:

 <p>Refresh</p>	<p>Refresh the tree view to pick up any changes</p>
 <p>Configure Destinations</p>	<p>Launches the Destination Explorer. From here you can configure the Destinations of the current K2 connect server, and set the Default Destination.</p> 
 <p>Settings</p>	<p>This allows you to specify the namespace for the .NET classes that are generated by the Service Object designer, and to beautify the names generated. This option is not used that often.</p>

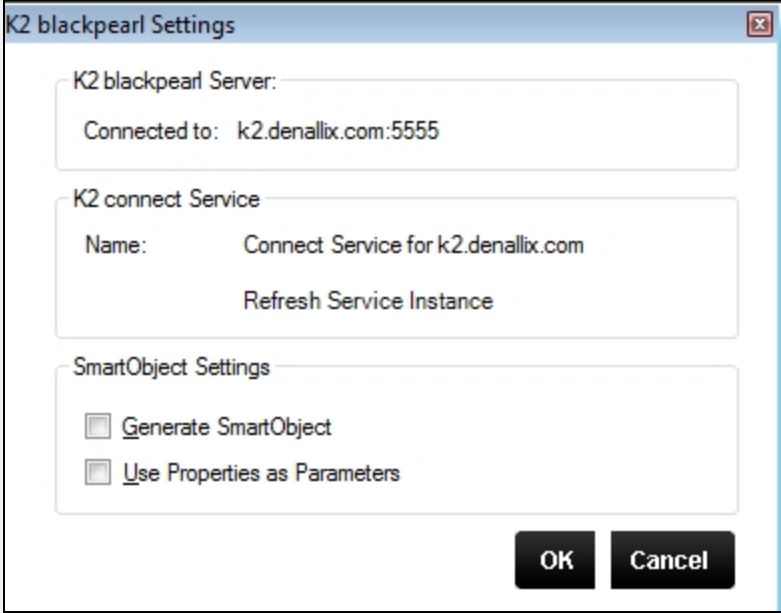


K2 blackpearl settings

This option allows you to refresh the Service Instance for the default destination remotely.


There is an option you can use to automatically **Generate SmartObject** when you publish the K2 connect Service Object. This is a nice time-saver, but we do not recommend using this option for production SmartObjects since you have no control over the naming for exposed Properties/Methods of the SmartObject.

The **Use Properties as Parameters** option will force users to enter the input properties for the SmartObject, which is useful when you want to be sure that users provide the required values. Again, this is primarily a time-saving tool and we recommend creating the SmartObjects manually. If you enable this option for a BAPI with many input properties, you could end up with a Service Object that requires a whole lot of input values before you can run the method, which may not be what you intended.



The dialog box titled "K2 blackpearl Settings" contains the following fields and options:

- K2 blackpearl Server:** A text box containing "Connected to: k2.denallix.com:5555".
- K2 connect Service:** A text box containing "Name: Connect Service for k2.denallix.com" and a button labeled "Refresh Service Instance".
- SmartObject Settings:** Two checkboxes: "Generate SmartObject" and "Use Properties as Parameters", both of which are currently unchecked.
- Buttons for "OK" and "Cancel" at the bottom right.



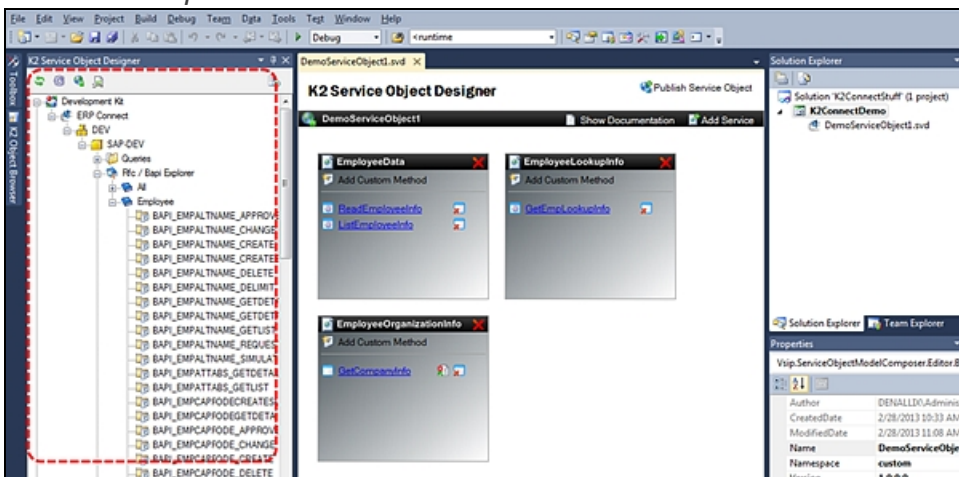
Stop/Start connect service

This option allows you to remotely start and stop the K2 connect service. When you publish and updated Service Object, the K2 connect service must be restarted to pick up the changes.

Destination explorer window

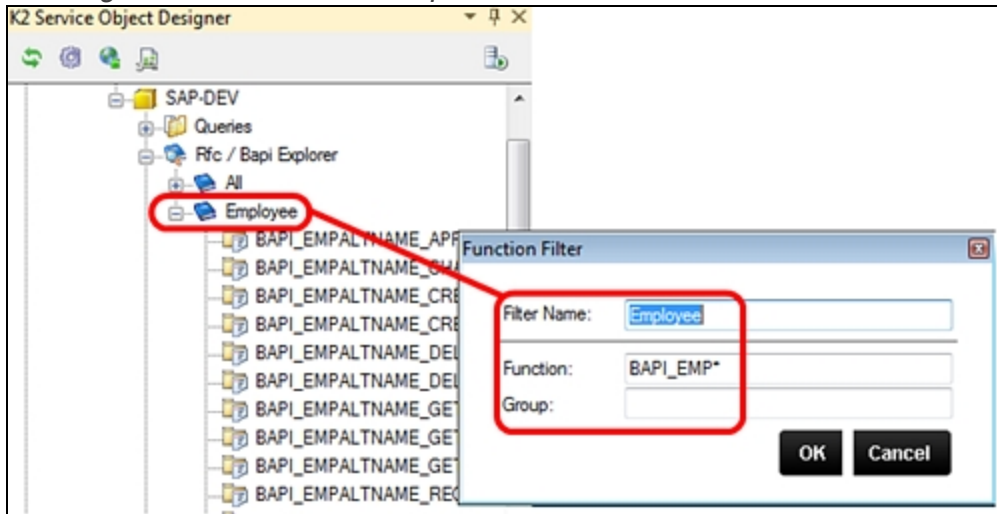
This section is where you connect to a K2 connect server, see a list of the Destinations configured for that server, and where you can explore the list of BAPI's in the SAP system targeted by that Destination. You can also launch the Test Cockpit from any of the BAPIs listed in the Service Object explorer.

Destination explorer



In the **Rfc/Bapi Explorer** section, you can define filters to limit the list of BAPIs returned. This is a good idea if you work with a particular set of BAPIs often or if you have a large set of BAPIs. When declaring a filter, use a * character as a wild card, like the example below where we return all the "EMPLOYEE" BAPIs.

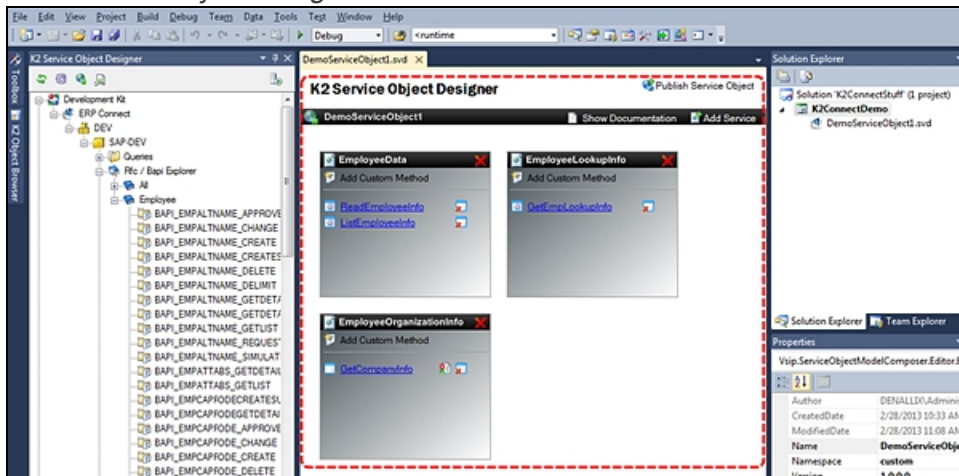
Creating a function filter for the Explorer window



Service Object Design Canvas

This is where you design your Service Object. You can drag and drop BAPIs from the Destination explorer window to auto-generate a Service Method for the BAPI, or you can manually define the Service methods and configure how the BAPI will be used. From here, you can also publish the Service Object to the K2 connect server.

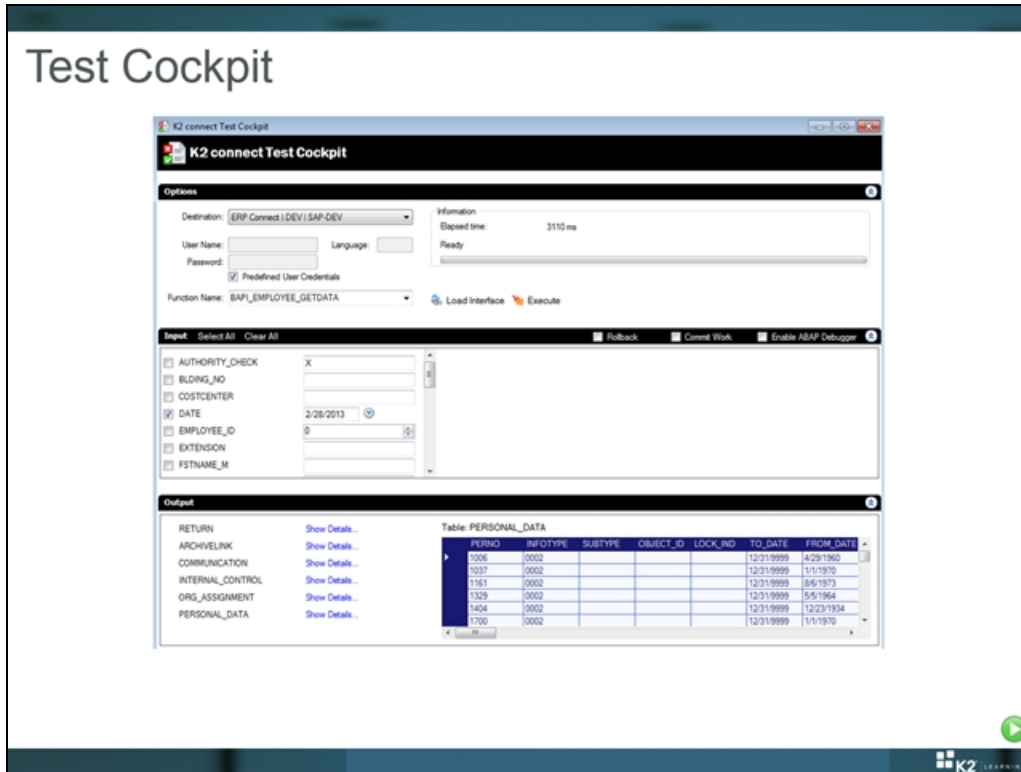
The Service Object Design Canvas



Solution Explorer

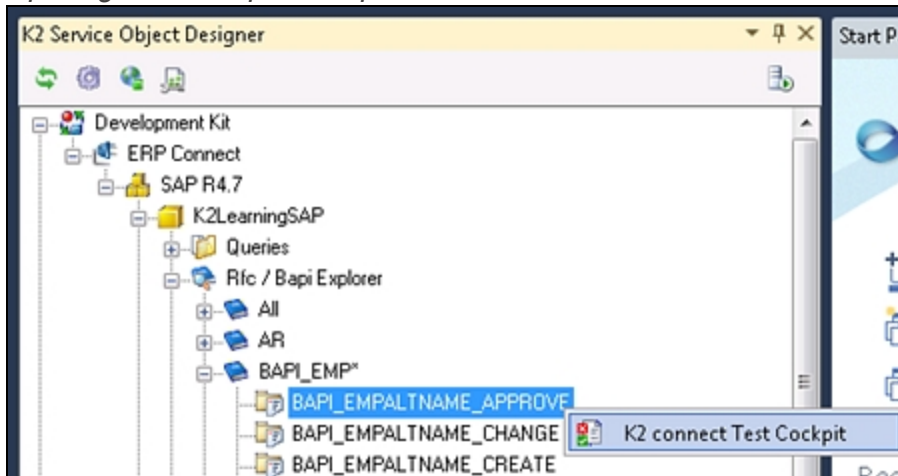
The solution explorer works just like any other Visual Studio project. While you can add any K2 artifacts to the project, it is recommended that you keep K2 Connect Service Objects and SmartObject in separate K2 projects, so that you can deploy the SmartObjects independently, or hand over the SmartObjects project to another developer without giving them access to the connect Service objects.

Test Cockpit



We briefly covered the Test Cockpit in the previous learning module, but since this is a valuable developer tool let's look at it again. The easiest way to launch the Test Cockpit is to select a BAPI in the Service Object Designer tree view, right-click the BAPI and select K2 connect Test Cockpit.

Opening Test Cockpit for a specific BAPI



You may need to provide credentials when running the tester, but if the Destination has embedded credentials, you can select the Predefined User Credentials option.

User Credentials in Test Cockpit

The screenshot shows the K2 connect Test Cockpit interface. The 'Options' section is highlighted with a red dashed box. It contains the following fields:

- Destination: ERP Connect | DEV | SAP-DEV
- User Name: [Empty]
- Password: [Empty]
- Predefined User Credentials:
- Function Name: BAPI_EMPLOYEE_GETDATA

Information: Elapsed time: 3110 ms, Ready. Buttons: Load Interface, Execute.

Input section:

- AUTHORITY_CHECK X
- BLDING_NO
- COSTCENTER
- DATE 2/28/2013
- EMPLOYEE_ID 0
- EXTENSION
- FSTNAME_M

Output section:

Table: PERSONAL_DATA

PERNO	INFOTYPE	SUBTYPE	OBJECT_ID	LOCK_IND	TO_DATE	FROM_DATE
1006	0002				12/31/9999	4/29/1960
1037	0002				12/31/9999	1/1/1970
1161	0002				12/31/9999	8/6/1973
1329	0002				12/31/9999	5/5/1964
1404	0002				12/31/9999	12/23/1934
1700	0002				12/31/9999	1/1/1970

Once the Test Cockpit has loaded, the first step is to use the **Load Interface** button to populate the **Input** and **Output** sections with the input parameters and return structures of the BAPI. You will likely need to provide at least one input parameter (SAP BAPI's usually requires at least a date input parameter). In the Input section, select the input properties you will provide values for, and un-check the rest.

Loading the BAPI interface

The screenshot shows the K2 connect Test Cockpit interface. The 'Load Interface' button in the 'Options' section is highlighted with a red dashed box. The 'Input' and 'Output' sections are also visible, showing the same data as the previous screenshot.

Next click the Execute button to run the method. If all went well, you should see Show Details next to each of the Output structures. You should always check the RETURN table first, since errors returned by SAP may be reported here. You can then look over the remaining tables and explore the data returned by the BAPI.

Executing a BAPI and reviewing the results

The screenshot displays the K2 connect Test Cockpit interface. The 'Options' section shows the destination set to 'ERP Connect | DEV | SAP-DEV' and the function name as 'BAPIEMPLOYEE_GETDATA'. The 'Execute' button is highlighted with a red dashed box. The 'Input' section contains several fields, with 'DATE' set to '2/28/2013' and 'AUTHORITY_CHECK' checked. The 'Output' section shows a table of 'PERSONAL_DATA' with columns for PERNO, INFO TYPE, SUBTYPE, OBJECT_ID, LOCK_IND, TO_DATE, and FROM_DATE.

Options

Destination: ERP Connect | DEV | SAP-DEV
 Information: Elapsed time: 3110ms
 Ready

User Name: Language:
 Password:
 Predefined User Credentials
 Function Name: BAPIEMPLOYEE_GETDATA

Input Select All Clear All Rollback Commit Work Enable ABAP Debugger

AUTHORITY_CHECK X
 BLDING_NO
 COSTCENTER
 DATE 2/28/2013
 EMPLOYEE_ID 0
 EXTENSION
 FSTNAME_M

Output

RETURN [Show Details...](#)
 ARCHIVELINK [Show Details...](#)
 COMMUNICATION [Show Details...](#)
 INTERNAL_CONTROL [Show Details...](#)
 ORG_ASSIGNMENT [Show Details...](#)
 PERSONAL_DATA [Show Details...](#)

Table: PERSONAL_DATA

PERNO	INFO TYPE	SUBTYPE	OBJECT_ID	LOCK_IND	TO_DATE	FROM_DATE
1006	0002				12/31/9999	4/28/1960
1037	0002				12/31/9999	1/1/1970
1161	0002				12/31/9999	8/5/1973
1329	0002				12/31/9999	5/5/1964
1404	0002				12/31/9999	12/23/1934
1700	0002				12/31/9999	1/1/1970

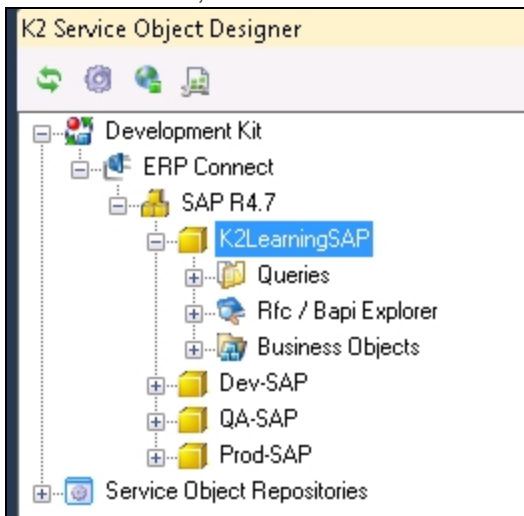
High-level steps to expose a SAP BAPI as a SmartObject

High-level steps to expose a SAP BAPI as a SmartObject

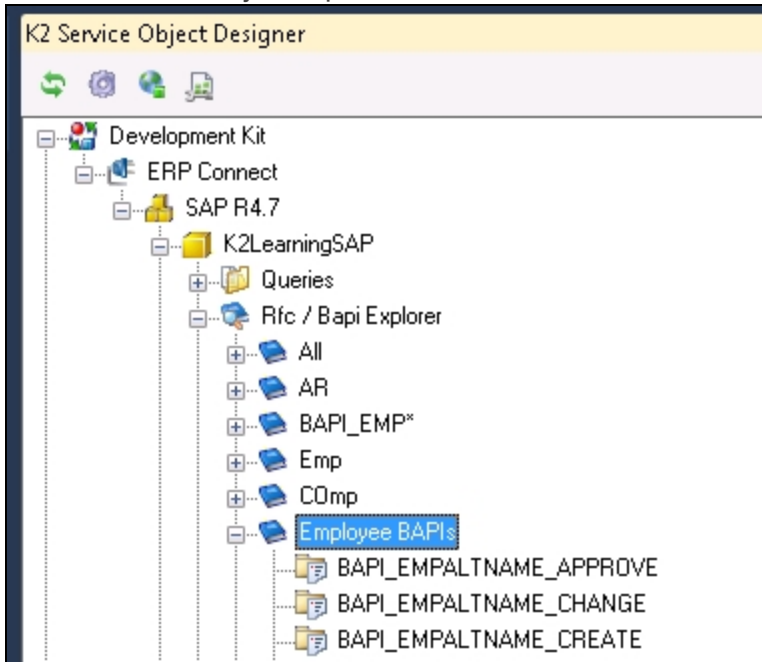
1. In Visual Studio, connect to a K2 connect server and select a target Destination
2. Use the Service Object explorer to discover the available BAPIs
3. Use the Test Cockpit to test and discover a particular BAPI
4. Create a new connect Service Object
 - a) Drag the BAPI into the Design Canvas and/or
 - b) Create custom methods
5. Configure parameters, properties and structures
6. Publish the connect Service Object
 - a) connect service will be restarted
 - b) The Default Destination's Service Instance is refreshed
 - c) Note: if updating an existing Service Object, you may need to restart the blackpearl service
7. Create a new SmartObject in Advanced mode
8. Use the SmartObject in a Workflow, User Interface or Report.

Before we start looking at the development tools in more detail, let's look at the general process you will follow to expose a SAP BAPI as a K2 SmartObject.

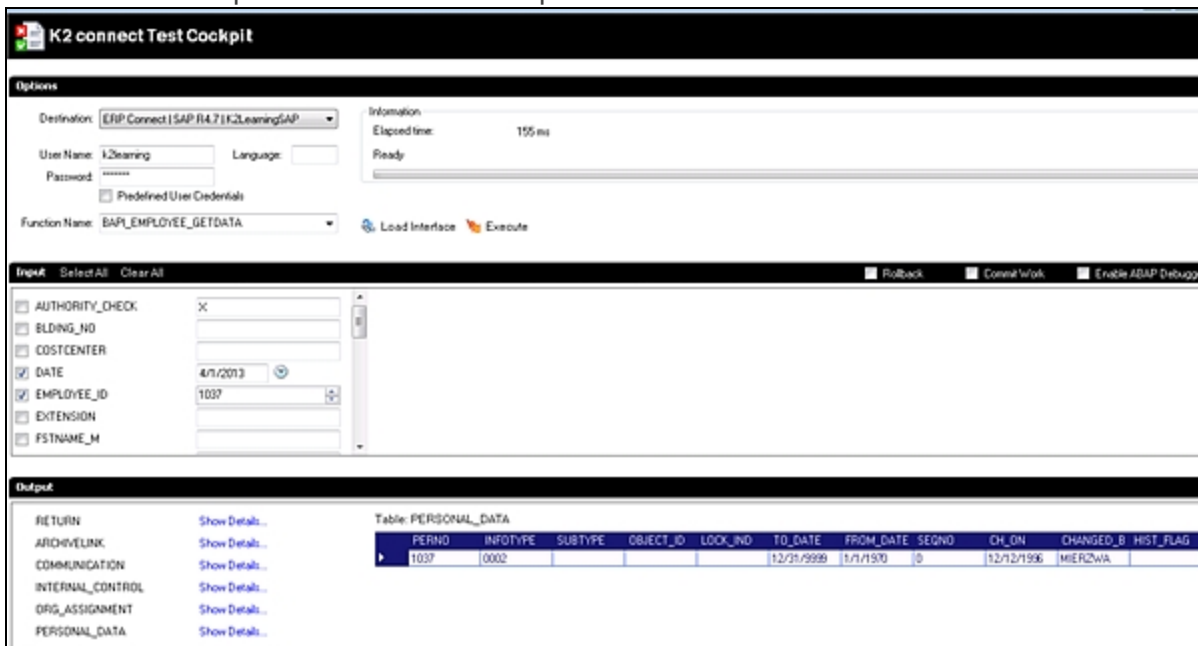
1. In Visual Studio, connect to a K2 connect server and select a target Destination.



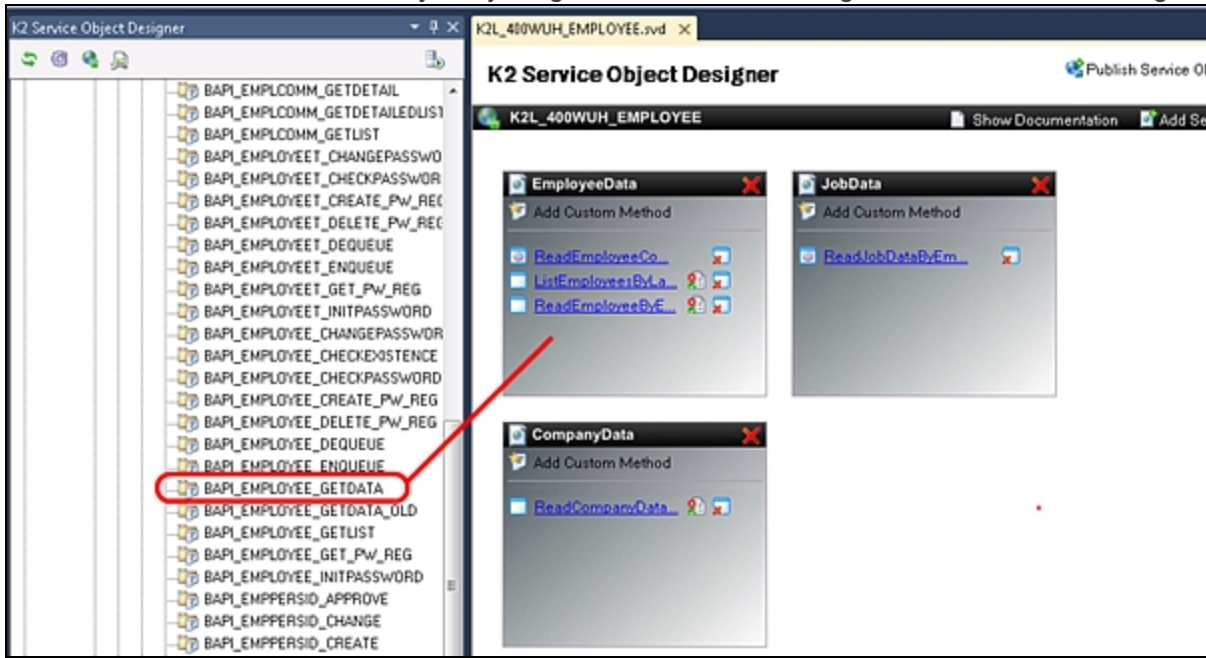
- Use the Service Object explorer to discover the available BAPIs.



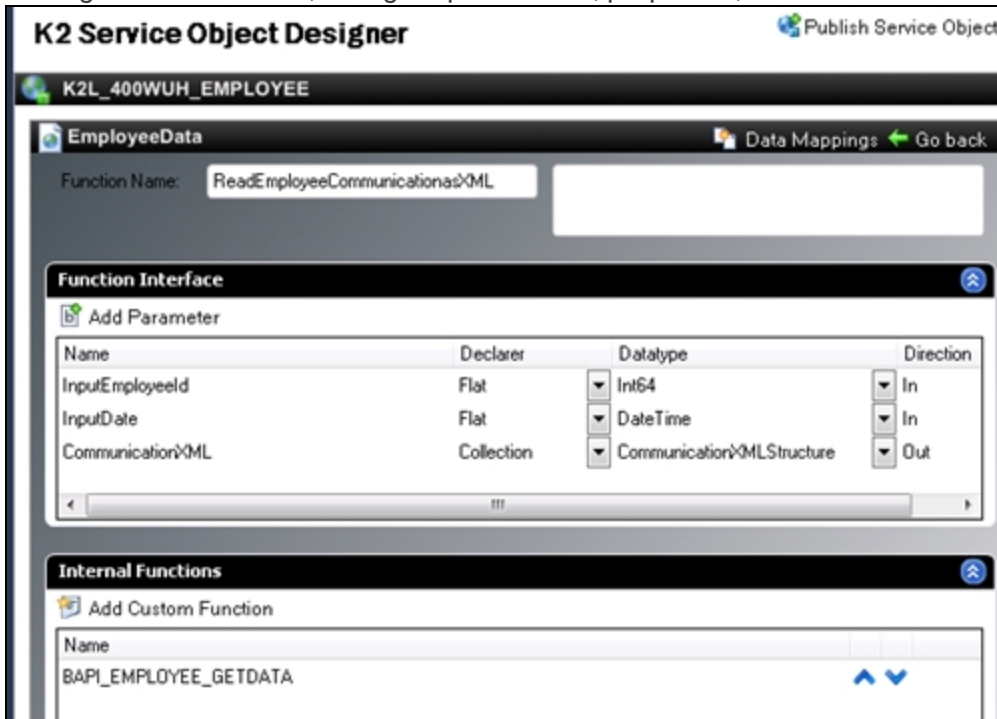
- Use the Test Cockpit to test and discover a particular BAPI.



4. Create a new connect Service Object by drag the BAPI into the Design Canvas and/or creating custom methods.

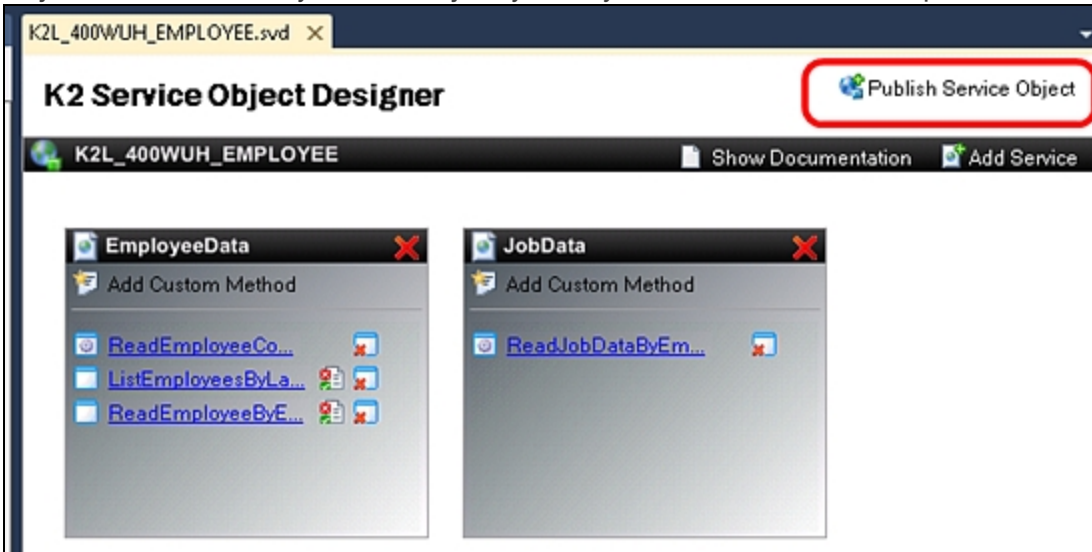


5. If using custom Methods, configure parameters, properties, and structures.

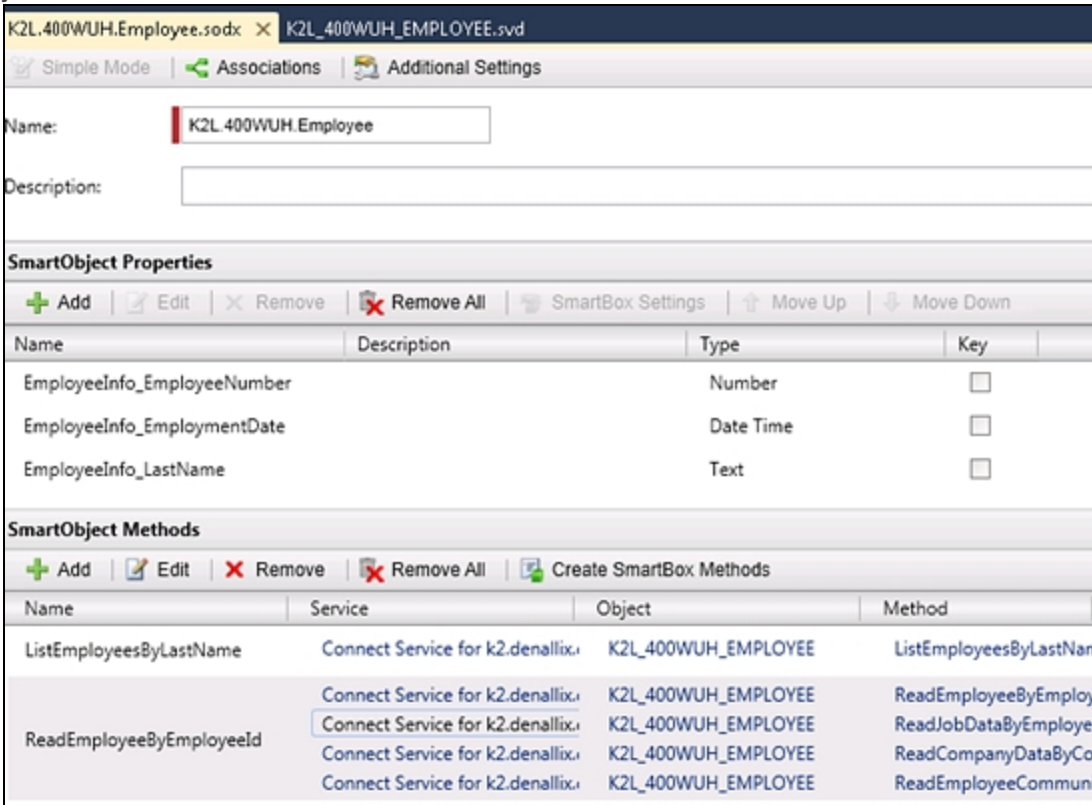


6. Publish the connect Service Object. During this process, the connect service will be restarted and the Default Destination's Service Instance is refreshed. Note: if you are updating an existing Service Object and that Service

Object has been used by a SmartObject, you may need to restart the blackpearl service.

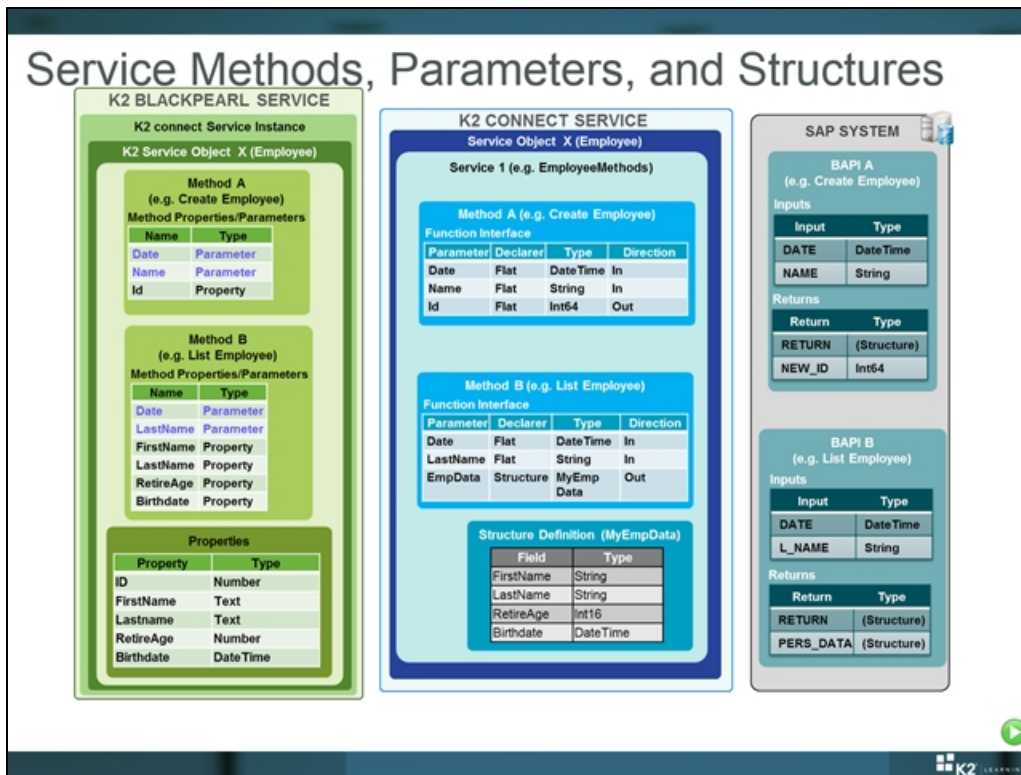


7. Create a new SmartObject in Advanced mode and use or combine the Service Object methods in your SmartObject.



8. Once the SmartObject is published, use the SmartObject in a Workflow, user interface, or Report. You can use SAP-backed SmartObject just like you would use any other SmartObject.

Service Methods, Parameters and Structures

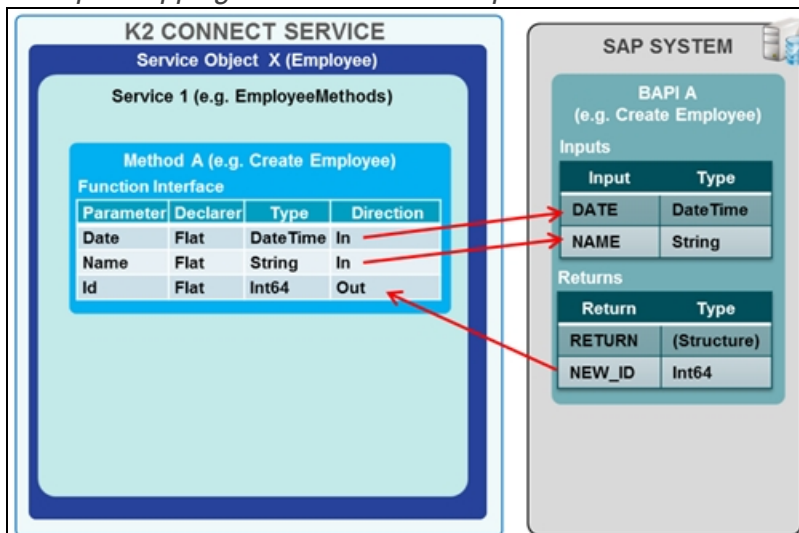


In the previous learning module, you created a simple connect Service Object by just dragging a specific BAPI onto the design canvas. While this is an easy and quick way to create Service Objects, it is likely that you will need to have finer control over the Service methods and the parameters passed in to SAP and returned from SAP. Let's take a moment to explain how BAPI input and output parameters are mapped to connect Service Object properties and method parameters, so that you can choose the best approach for your requirements.

Consider the diagram below, which illustrates a simple BAPI (**BAPI A**). This BAPI accepts two primitive data types as input (a date and a string) and it returns two parameters: **NEW_ID** is an integer returned by SAP and **RETURNS** (which contains the result of the BAPI from SAP - we are not interested in showing this result in the SmartObject).

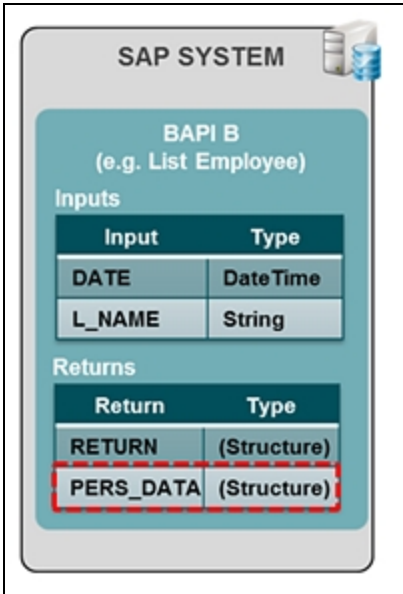
When we create a connect Service Object method for the SAP BAPI, the method's *Function Interface* describes how the properties are mapped to the BAPI's input and output parameters.

A simple mapping between SAP BAPI parameters and the Service Method Function Interface



A more realistic scenario is that the SAP BAPI returns complex structures instead of primitive data types. Consider the diagram below: here, we have a BAPI for retrieving employee data from SAP, but the Employee Data (PERS_DATA) itself is a structure or table of data, as you can see from the Test Cockpit output.

BAPI with complex return data



Exploring the return data structure in test Cockpit

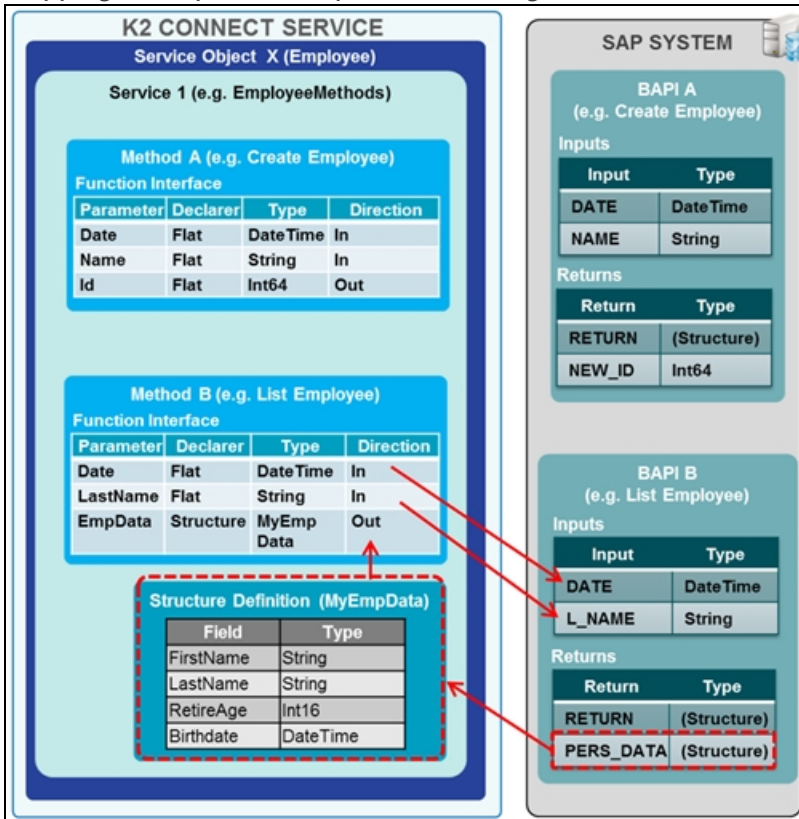
The screenshot shows the 'Output' section of the SAP Test Cockpit. On the left, there is a list of return objects with 'Show Details...' links. The 'PERS_DATA' object is highlighted with a red circle. On the right, a table titled 'Table: PERS_DATA' is displayed with the following columns: PERNO, INFOTYPE, LOCK_IND, TO_DATE, FROM_DATE, SEQNO, and CH_ON. The table contains 15 rows of employee data.

PERNO	INFOTYPE	LOCK_IND	TO_DATE	FROM_DATE	SEQNO	CH_ON
1006	0002		12/31/9999	4/29/1960	0	10/13/2000
1037	0002		12/31/9999	1/1/1970	0	12/12/1996
1161	0002		12/31/9999	8/6/1973	0	7/25/2000
1329	0002		12/31/9999	5/5/1964	0	9/12/1996
1404	0002		12/31/9999	12/23/1934	0	5/3/2000
1700	0002		12/31/9999	1/1/1970	0	10/13/2000
1904	0002		12/31/9999	5/28/1966	0	2/20/1998
2120	0002		12/31/9999	2/23/1950	0	2/25/2003
2121	0002		12/31/9999	2/23/1951	0	5/20/2003
10155	0002		12/31/9999	1/1/1960	0	2/21/1997
10453	0002		12/31/9999	5/5/1965	0	11/6/2003
10966	0002		12/31/9999	1/11/1960	0	6/9/1999

Since the consumers of SmartObjects would not know how to interact with this complex structure, we need a mechanism to flatten the complex structure into primitive types that SmartObjects can understand. This is where Structure Definitions come into play. Using a Structure Definition, you can select which properties from the BAPI you want to expose in your Service Object. As part of this process, you can also rename the SAP BAPI properties into easier-to-read properties.

The diagram below illustrates this approach. The BAPI still has two simple input parameters (a date and a string). Since the **PERS_DATA** return is a complex type, the developer created a Structure Definition called **MyEmpData** which maps the data from the BAPI return table to an internal structure.

Mapping a complex return parameter using a Structure Definition



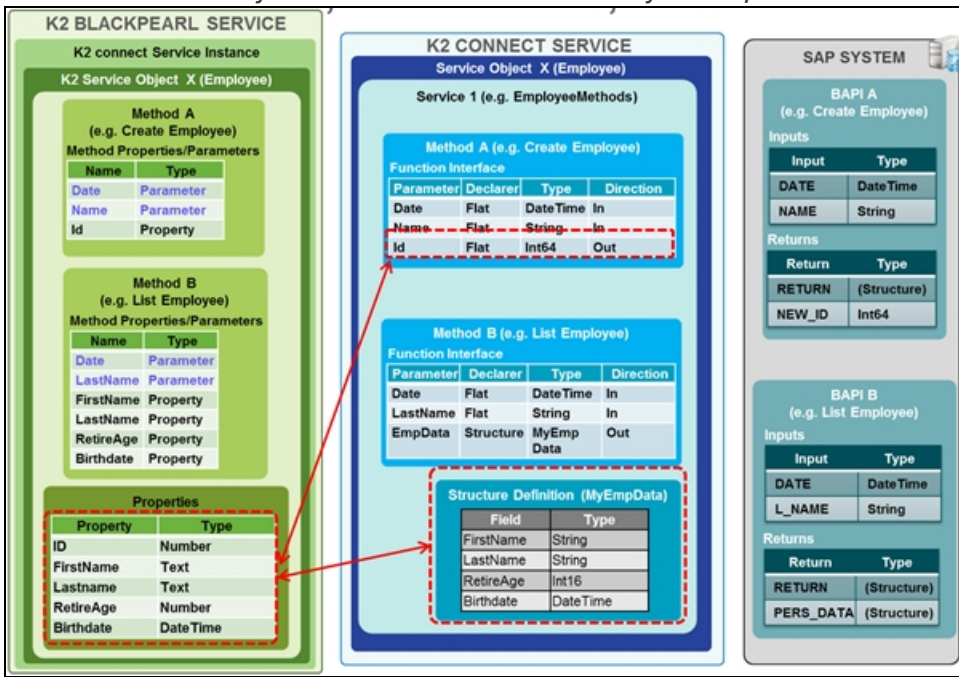
Note

Structure Definitions are stored on the Service level. This means you can reuse the same Structure Definition between the methods in a Service, but not outside of the Service. This is another reason why you'd want to keep methods that return the same data in the same Service: you can reuse the Structure Definition in the same Service.

The last piece of the puzzle is to understand how the Methods and Parameters defined in the connect Service Object surface to a K2 Service Object. Remember that K2 Service Objects have Properties and Methods, but Properties and Methods are defined separately. If a connect Service Object has multiple methods, each of the properties in each of the methods will be exposed as a property of the K2 Service Object.

Consider the diagram below. Notice that the K2 Service Object has two Methods which map to the methods defined for the connect Service Object, and the K2 Service Object's methods have input parameters for the "In" parameters defined for the connect Service Object's methods. Notice also that the Service Object has a shared set of Properties, which include all the properties returned by all the methods in the connect Service Object.

Connect Service Object methods vs. K2 Service Object Properties and Methods



This might all appear very abstract at the moment, but after completing the lab exercise you will see how this mapping is defined and ultimately exposed.

Custom Service Methods

Custom Service Methods

- Recommended approach
- Allows greater control of returned values
- Allows “friendly names” for SAP properties and methods
- Can combine properties from different Return structures
- Return properties as XML
- Manually define data mappings between connect parameters and BAPI parameters
- Write custom code to transform data types between connect and SAP
- Call multiple BAPIs in sequence (e.g. when writing data to SAP)
- Write custom code in functions



While the drag-and-drop approach is easy to use when creating connect Service Objects, in reality you will often use the Custom Service Methods approach instead. So why would one go to the trouble of defining a custom Service Method when you can just drag and drop a BAPI into the Service Object design canvas and be done with it? Well, there are quite a few benefits when you use custom service methods, and in some scenarios you will have to define a custom service method, since the drag-and-drop approach might not support what you are trying to achieve.

Using custom Service Methods is the recommended approach, since it allows you much greater control over the values that are returned from the BAPI while allowing you to define user-friendly names for the Service Object methods and properties. This will make it much easier to design K2 SmartObjects that will use the Service methods. You can also use this approach to select different properties from different return structures in a single BAPI call and return specific properties as native XML data rather than flattening the structure to primitive data types.

In more advanced scenarios, you can manually define mappings and transformations of parameters between the connect Service Object and the underlying SAP BAPI, you can call multiple BAPIs in a predefined sequence (this is often required when you want to write data to SAP), or you can write completely custom code in custom functions.

Realistically speaking, you will probably use manual Service Methods in the majority of your connect Service Objects except for very basic BAPI interactions, so you should become very familiar with this process.

Multiple Services vs. Multiple Service Objects

A connect Service Object may contain many Services and each Service could contain multiple methods that point to SAP BAPIs. While you could define a single Service and add all the BAPIs into this one service, it is a good idea to structure the Services and their enclosed methods logically.

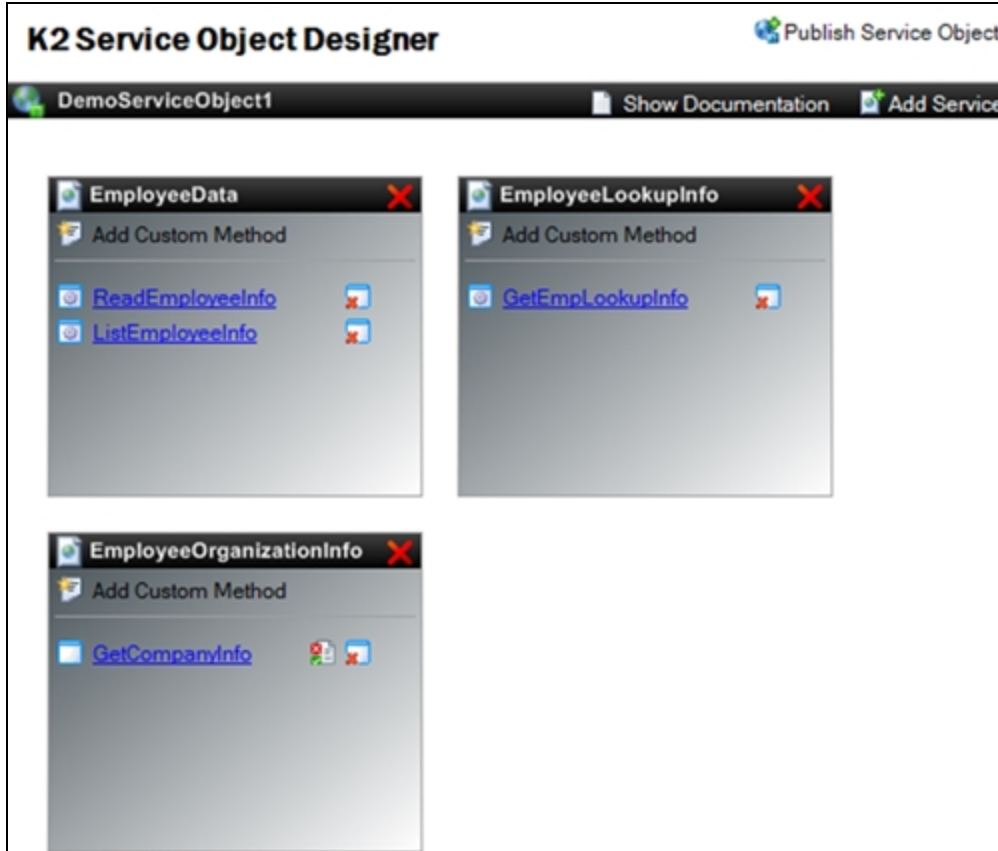
Consider the sample Service object below. This service object contains three Services (**EmployeeData**, **EmployeeLookupInfo** and **EmployeeOrganizationInfo**). Each service returns different data structures, but they are all related to an employee in some way.

The **EmployeeData** service contains two methods (**ReadEmployeeInfo** and **ListEmployeeInfo**). Both services return the same data structures, but one returns a single record (Read) and the other returns a list of records (List). Since these methods return the same data, we grouped them into the same Service. Since they both use the same data struc-

ture in the background, we can re-use a custom Data Structure definition because the methods are located in the same Service.

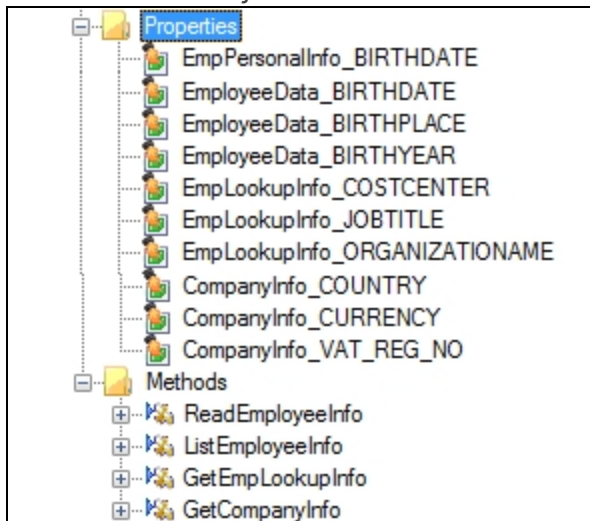
The other two services (**EmployeeLookupInfo** and **EmployeeOrganizationInfo**) call BAPIs that return lookup information for an Employee. Since these methods return different data structures, we placed them into separate services, but since they relate to an employee (and only to an Employee - they are not used by any other SmartObjects), we added them to the Employee service object.

Sample Service Object with multiple Services



When you deploy the Service Object as a SmartObject, all the methods and the properties returned by the methods will be flattened, as shown in the example below. Therefore, the way you structure Methods and Services in the Service Object designer doesn't really surface anywhere, but it is a nice approach to use to separate the data structures between the various methods used in the Service Object or re-use data structures.

The K2 Service Object version of the Service Object



When adding multiple Service Objects, consider how you would balance the ability to use different BAPIs in the same Service Object versus creating a separate Service Object for BAPIs that return different data. A rule of thumb you can use is to keep related data together in the same SmartObject. If any data is required by another SmartObject, you may want to create a separate Service Object. Here is an example: suppose you had a BAPI that returns departmental information. This data could relate to an Employee, but also to Payroll or Finance functions, so it is probably a good idea to create a separate *DepartmentInfo* Service Object so that you can use that same Service Object in other SmartObjects.

Using XML properties instead of Structures

Using XML properties instead of Structures

Function Name: BAPI_EMPLOYEE_GETDATA

Input: Select All Clear All

Output: RETURN, ARCHIVELINK, COMMUNICATION, INTERNAL_CONTROL, ORG_ASSIGNMENT, PERSONAL_DATA

PERNO	INFOTYPE	LOCK_IND	TO_DATE	FROM_DATE	SEQNO	CH_ON
1037	0105		12/31/9999	1/1/2004	0	1/15/2004
1037	0105		12/31/9999	1/1/2000	0	9/25/2000

Collection of items

When you need to return complex Structures from the SAP BAPI you have a choice whether to flatten those structures to primitive data type properties, or to return the property as an XML document. Sometimes you might need the BAPI property returned as XML so that you can process the XML yourself, or perhaps the property contains a collection of items which is difficult to represent as a single property. In these cases, you can choose to return the complex structure as an XML property instead of a Structure Definition.

To return a complex type as an XML property, select the **XML Property** checkbox as shown in the screenshot below. When you select this option, the generated K2 Service Object will have an XML property which contains the properties you selected, rather than separate flattened properties for each property in the structure.

This approach is commonly used when you want to pass or return a collection of values as a single property from a SAP BAPI. Note, that you may need to write some code in the consuming application to handle the XML processing for the property.

The screenshots below show an example of how a collection of items could be returned as an XML property.

The COMMUNICATION return is a collection of items

Function Name: BAPI_EMPLOYEE_GETDATA

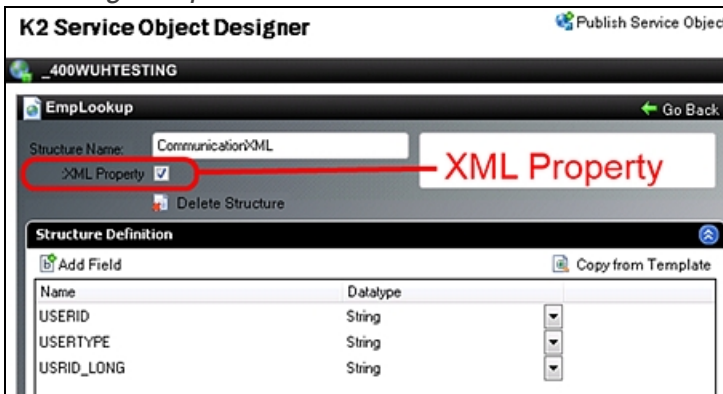
Input: Select All Clear All

Output: RETURN, ARCHIVELINK, COMMUNICATION, INTERNAL_CONTROL, ORG_ASSIGNMENT, PERSONAL_DATA

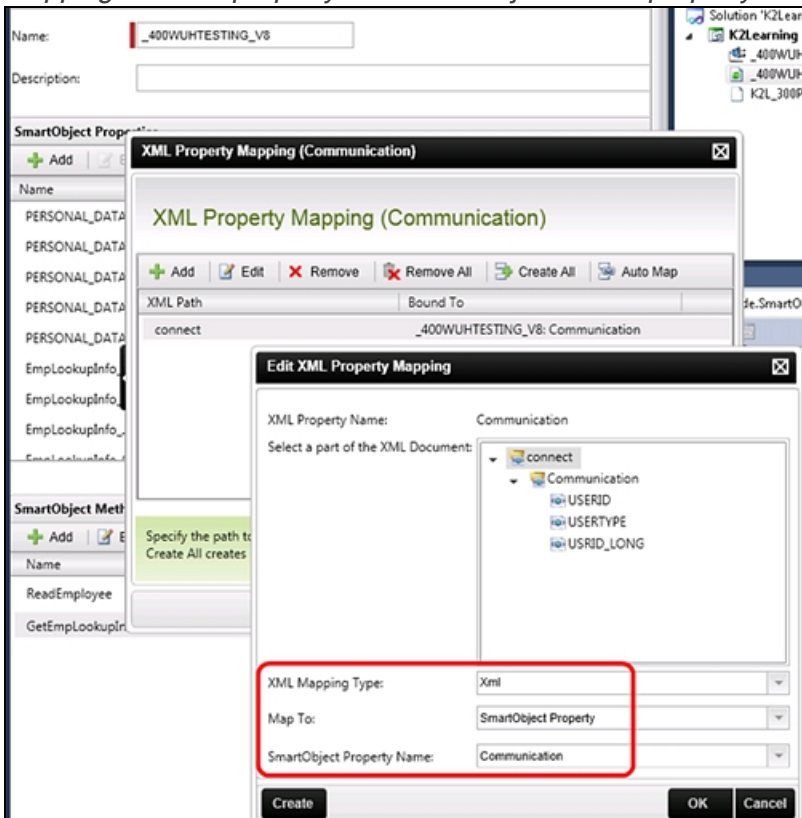
PERNO	INFOTYPE	LOCK_IND	TO_DATE	FROM_DATE	SEQNO	CH_ON
1037	0105		12/31/9999	1/1/2004	0	1/15/2004
1037	0105		12/31/9999	1/1/2000	0	9/25/2000

Collection of items

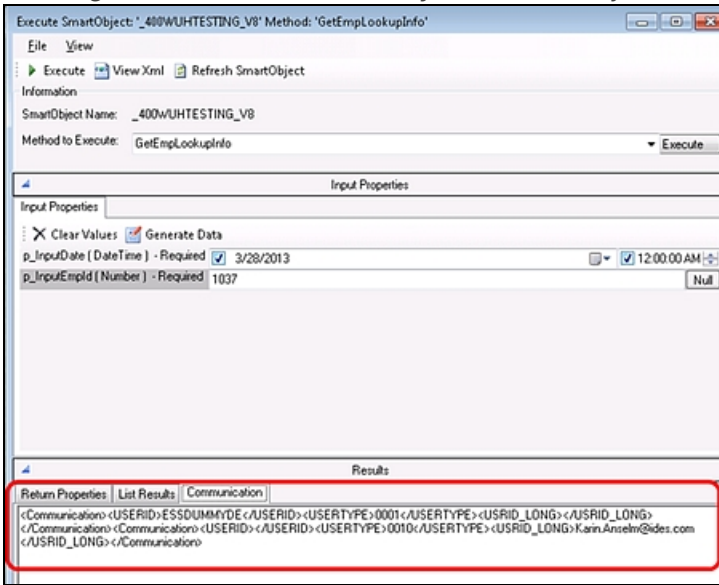
Selecting the option to return the COMMUNICATION structure as an XML property of the Service Object



Mapping the XML property to a SmartObject memo property



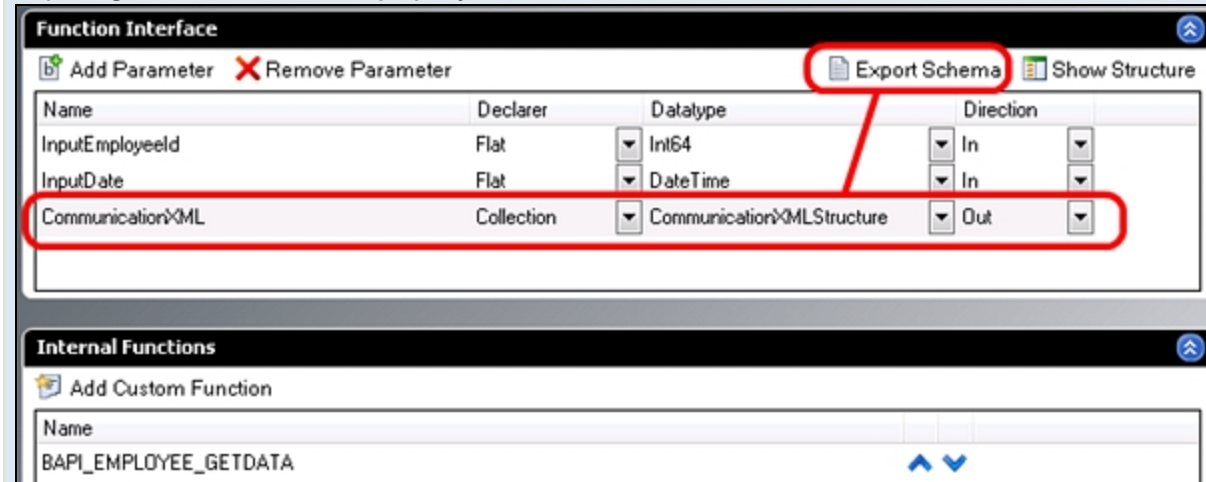
Viewing the result in the SmartObject Tester utility



Note

Note that you can easily generate an .xsd for the XML property, which will make it easier for you to interact with the Property in other applications like C# code or workflows.

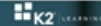
Exporting the schema of a XML property



What happens when you publish a Service Object?

What happens when you publish a Service object?

- When publishing a Service Object with the K2 Service Object Designer, the following happens:
 1. The .svd file is compiled to C# classes
 - Might get compilation errors if class names are invalid
 - .svd definition is published to the **default Destination**
 2. The K2 **connect Server Service** will be **restarted**
 - This compiles new proxy files hosted by K2 connect service
 - Proxy files describe the Methods in the Service Object
 3. The **default K2 connect Service Instance** will be **refreshed**
 - Service Instance discovers the updated object definition in K2 connect service
 - Creates the proxy DLL files for use by K2 blackpearl
- If you have made changes to a Service Object that is currently being used:
 - K2 blackpearl holds the proxy files in memory for faster execution, but only if the SmartObject was used previously.
 - You may need to restart the K2 blackpearl Server service to load the new proxy files



It is important to understand the mechanics behind the Service Object publish process, since there are some nuances that might affect what you need to do when publishing Service Object definitions to a K2 connect server.

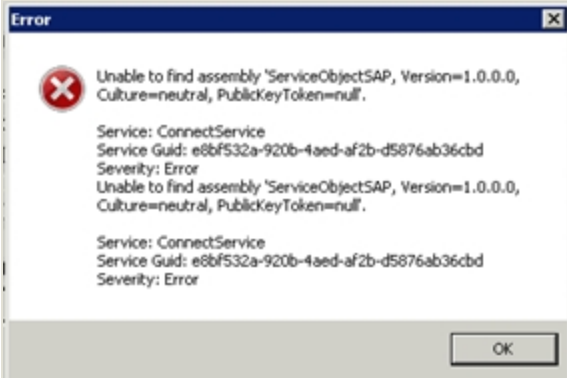
When you use the **Publish Service Object** link in the Visual Studio Service Object designer, the first step is that Visual Studio will compile the .svd file to C# classes. At this stage, you may see errors if the connect Service Object properties do not follow standard .NET naming restrictions, or if something is wrong inside custom code that you may have written. If the connect Service Object compiles successfully, the connect Service Object is published to a K2 connect server. The K2 connect service is restarted (or must be manually restarted if the .svd was published manually without using Visual Studio) so that the new proxy classes for the updated connect Service Object can be generated and loaded.

Once the connect Service is restarted, the Service Instance for the K2 connect server is refreshed. (If there are multiple connect service instances defined on the same K2 blackpearl server, you may need to refresh the Service instance for your selected Destination manually since the built-in Publish process only refreshes the default Service Instance automatically). As part of this process, K2 blackpearl will interrogate the K2 connect server, "discover" the updated connect Service Object definition and then update the K2 Service Objects for the Service Instance. At this point, a SmartObject developer can create a new SmartObject that uses the new definition, or they may modify an existing SmartObject to use the new methods/properties that are available.

Note

If you have made changes to a Service Object that is currently used by a SmartObject and that SmartObject has been executed by K2 blackpearl before, note that K2 blackpearl holds the proxy files for the Service Object in memory for faster execution.

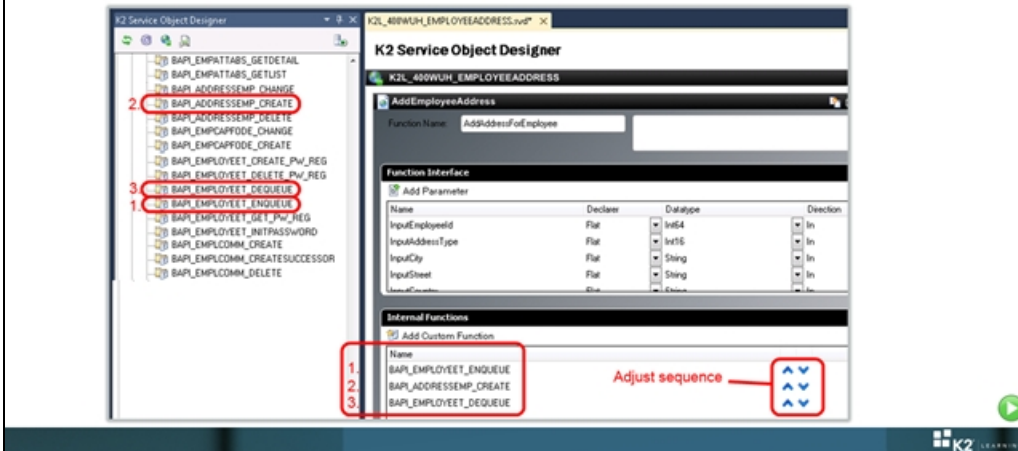
You may need to restart the K2 blackpearl Server service to load the new proxy files, otherwise you might see errors like "Unable to find assembly..." like the example below:



Writing data to SAP

Writing data to SAP

- You will need to create a custom Service Method
 - Call BAPI_aaa_ENQUEUE with Key ID to “Lock” the record
 - Call BAPI_bbb with input values to create/update data
 - Call BAPI_aaa_DEQUEUE with Key ID to “Unlock” record
- Example: Adding an Employee address

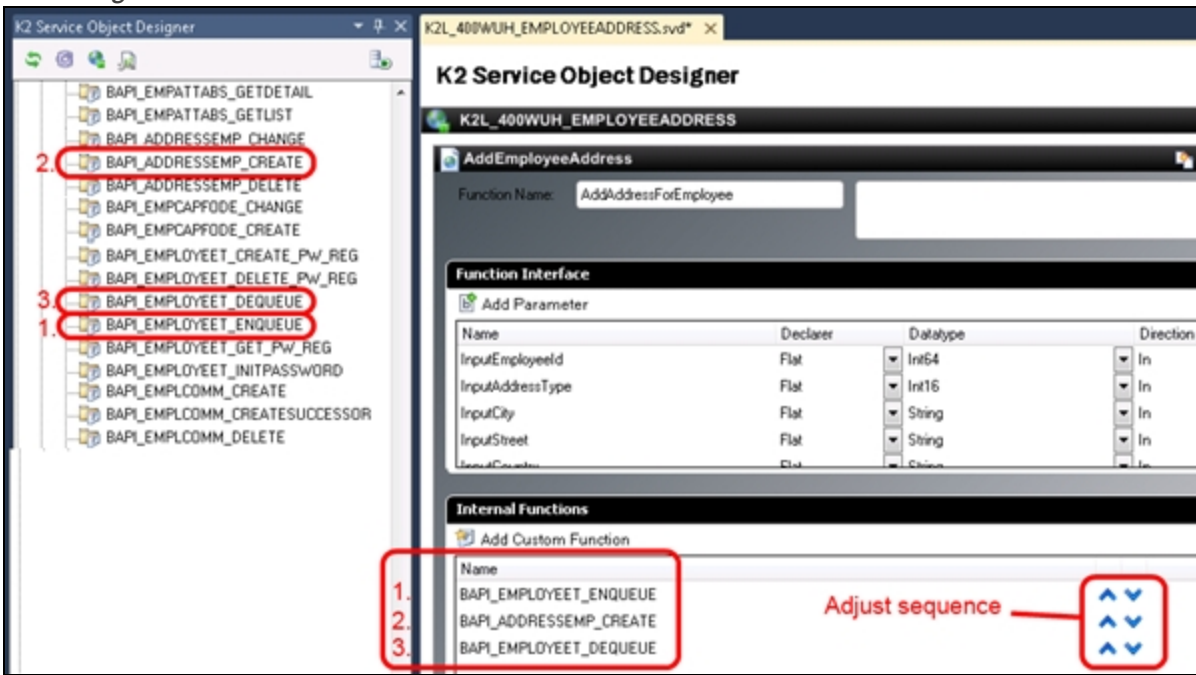


When you need to write data to SAP, be aware that you may need to chain several BAPIs together to update data. SAP has BAPIs that are used to lock a record so that it can be updated and other BAPIs that are used to unlock a record after changes have been applied.

Normally, the procedure is to call the ENQUEUE BAPI for the object that you are about to update, then call the BAPI that actually does the update, and then call the DEQUEUE BAPI for the same object to commit the changed data.

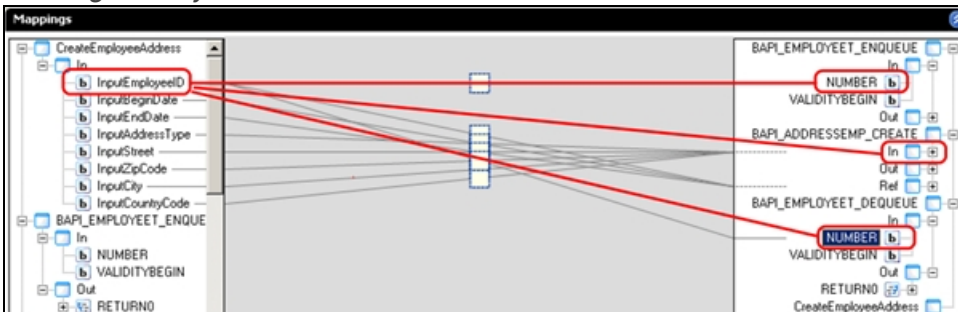
Consider the screenshot below, where we are adding a new address for an employee with the BAPI_EMPADDRESS_CREATE BAPI. Notice that we are wrapping this BAPI between the BAPI_EMPLOYEE_ENQUEUE and BAPI_EMPLOYEE_DEQUEUE BAPIs so that the data can be written to the EMPLOYEE object.

"Chaining" BAPI calls to write data to SAP



When you use these BAPIs, also note that you would normally have to pass the ID of the record being updated to each of the BAPIs. This is achieved by using the mapping screen to reuse the EmployeeID input parameter in all of the chained BAPI calls, as shown below:

Passing the Key ID into each BAPI call



Custom Transformation Mapping Code

Custom Transformation Mapping code (1/2)

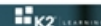
- Use custom code in data mappings to transform input and output parameters in the mapping screen
- Method will have the SAP input/output parameter and the connect input/output parameter as method parameters
- Write c# code to manipulate the parameters as required
- Often used for returning SAP documents as SmartObject File properties, since SAP's binary format must be converted to a base-64 encoded string
- Example in documentation:
Development & User Guide > Filtering > Adding & Removing Filters > Data Mapping > Using K2 Connect Transformation mapping to return SAP binary data to SmartObjects



Custom Transformation Mapping code (2/2)

The screenshot shows two windows from the K2 Connect interface. The top window, titled 'Service2', displays a 'Mappings' tree. A mapping is shown between 'PTX_CONTENT' (SAP: Standard binary format) and 'FileContent' (Byte[]: Base-64 Binary Format). The bottom window, titled 'Transformation', shows the 'Net Code' editor with the following C# code:

```
private void Transform (custom.K2L_400F0UH_EMPLOYEEADDRESS.Types.Service2.CollectionsPLN_DMS_CONTENT PTX_CONTENT,
ref System.Byte[] FileContent)
{
    byte[] bytes;
    if (PTX_CONTENT.Length > 0)
    {
        long length = 0;
        System.IO.MemoryStream ws = new System.IO.MemoryStream();
        System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bf =
            new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
        bf.Serialize(ws, PTX_CONTENT);
        length = ws.Length;
        bytes = ws.GetBuffer();
        string encodedData = bytes.Length + ":";
        File = Convert.ToBase64String(bytes, 0, bytes.Length, Base64FormattingOptions.None);
    }
}
```



In some situations, you may need to write code in the transformation mappings to convert a SAP property/parameter into a K2 connect Service Object property/parameter or vice-versa. The most common reason for a custom transformation mapping is when the native SAP data type does not match any of the available data types for a K2 SmartObject, even when the SAP data type has been flattened. In practice, this most often happens with documents stored in SAP, since SAP uses native binary encoding while a SmartObject File property is a base-64 encoded string.

To achieve the mapping from the SAP data type to a SmartObject data type, you can write C# code in a transformation mapping. The method will have the SAP parameter and the connect Service method parameter as inputs, and you can write the code that converts one to the other.

There is an example of this approach in the K2 connect product documentation, at *Development & User Guide > Filtering > Adding & Removing Filters > Data Mapping > Using K2 Connect Transformation mapping to return SAP binary data to SmartObjects*.

Consider the diagram below. Here, we are trying to return a document object from SAP (represented by the **PTX_CONTENT** field on the left side) to a **FileContent** Service Object property on the right. To convert the native SAP binary data type to a base64-encoded string, we wrote a C# method to perform the conversion in the transformation mapping block.

Using code to convert a SAP data type to a connect Data Type

The screenshot displays the K2 Connect interface. The top window, titled "Service2", shows a "Mappings" view. On the left, the "DMS_DOCUMENTS_TO_OBJECT" data type is expanded, showing fields like "PTX_CONTENT" and "PTX_ID_DOCS". On the right, the "DMS_DOCUMENTS_TO_OBJECT" data type is also expanded, showing fields like "FileContent". A red box highlights the "PTX_CONTENT" field on the left and the "FileContent" field on the right. A red arrow points from the "PTX_CONTENT" field to the "FileContent" field. Below the "Mappings" window, a "Transformation" window is open, showing a "Net Code" block. The code is as follows:

```
private void Transform (custom.K2L_400WUH_EMPLOYEEADDRESS.Types.Service2.CollectionPLM_DMS_CONTENT PTX_CONTENT,
ref System.Byte[] FileContent)
{
    byte[] bytes;
    if (PTX_CONTENT.Length > 0)
    {
        long length = 0;
        System.IO.MemoryStream ms = new System.IO.MemoryStream();
        System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bf =
            new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
        bf.Serialize(ms, PTX_CONTENT);
        length = ms.Length;
        bytes = ms.GetBuffer();
        string encodedData = bytes.Length + ":";
        = Convert.ToBase64String(bytes, 0, bytes.Length, Base64FormattingOptions.None);
        File = encodedData;
    }
}
```

Red annotations in the image include: "SAP: Standard binary format" pointing to the "PTX_CONTENT" field; "Byte[]: Base-64 Binary Format" pointing to the "FileContent" field; and a red box around the "Net Code" block.

Deploying to other servers/environments

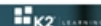
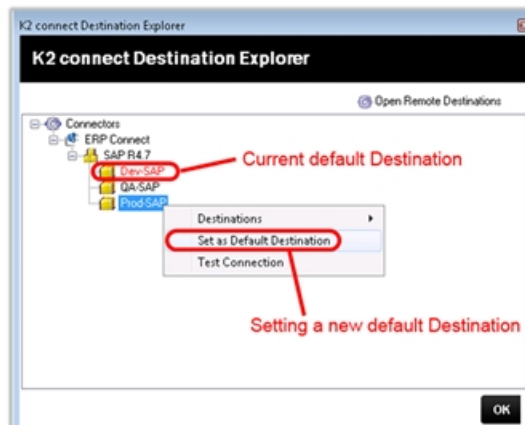
Deploying to other servers/environments (1/2)

- Important: target SAP systems MUST expose the same BAPIs
- Important: connect Service Instances must have the same GUIDs
- Connect Service Objects are *published* to default destination on a K2 connect Server
- K2 SmartObjects are *deployed* to a K2 blackpearl server and are associated with a Service Instance
- Destinations must already be configured on target K2 connect servers
- Publish procedure depends on how Destinations are configured
 - Connect servers can have multiple destinations, and/or
 - Your environment may have separate K2 connect Servers
- If a connect Server has multiple Destinations
 1. Set Default destination
 2. Publish the Service Object
 3. Refresh connect Service Instance (if needed)
- If publishing to a different connect server
 1. Open remote destinations
 2. Set default destination (if needed)
 3. Publish the Service Object
 4. Refresh connect Service Instance (if needed)
- Deploy updated SmartObjects and restart blackpearl service (if needed)



Deploying to other servers/environments (2/2)

- You can set the default Destination in connect Administration console or Visual Studio
- Current Default Destination is **red**
- Right-click another Destination and select **Set as Default Destination**



Most enterprise deployments of K2 and SAP will have multiple environments (for example DEV, QA, and PROD). While you will most likely create your connect Service Objects in the DEV environment, at some point you will want to deploy your artifacts to the QA and Production environments.

First, the two most important requirements: the target SAP environments MUST expose the same BAPIs, and if you are deploying K2 SmartObjects, the connect Service Instances for the SmartObjects must have the same Service Instance

GUIDs. We also assume that the Destinations for the SAP systems have already been set up on each of your target K2 connect/K2 blackpearl servers. (You may want to refer back to the topic [Multiple Destinations](#) in the module **K2 connect Configuration and Administration** for a refresher on how SAP systems map to K2 connect Destinations).

Note

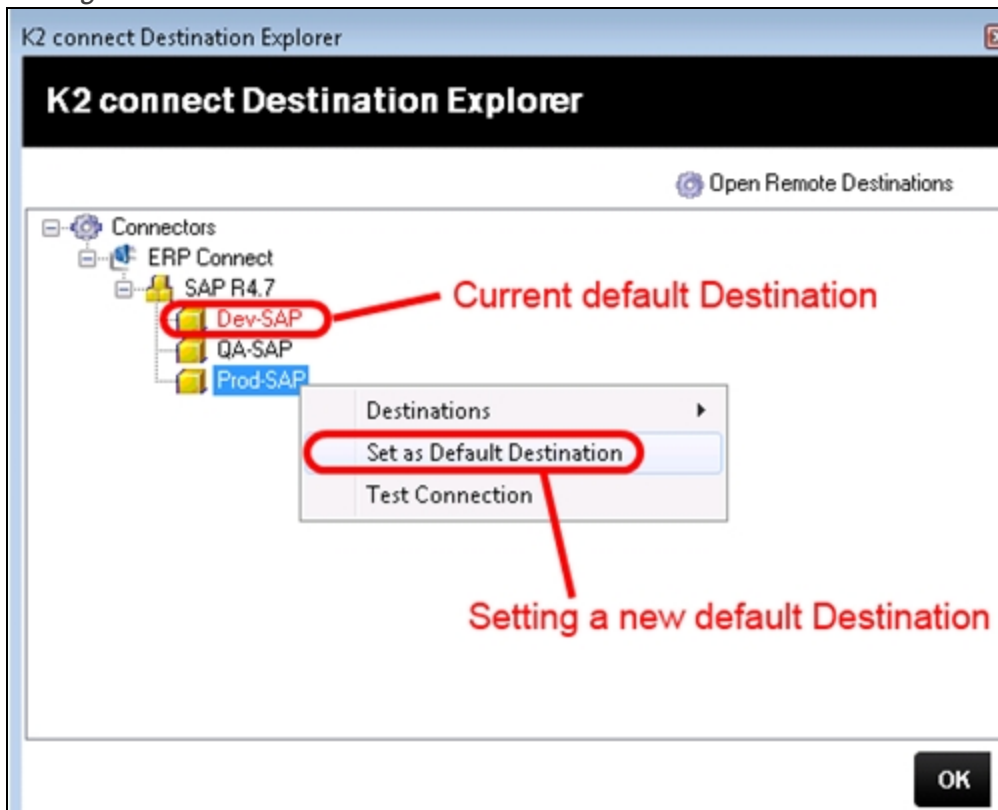
Remember that K2 connect Service Objects are published to a K2 connect server, and are associated with the Default Destination on that K2 connect Server. If needed, you may need to change the Default Destination before publishing the connect Service Object.

K2 SmartObjects are deployed to a K2 blackpearl server, and are associated with a specific connect Service Instance. That's why the Service Instance GUIDs for connect Service Instances must match between the different K2 environments.

The publish procedure depends on the way that Destinations were configured on your K2 connect systems. In all cases, note that when a K2 connect Server has multiple Destinations configured, you may first need to change the Default Destination on the K2 connect Service to the destination that your Service Object uses. This is a simple procedure, and you can use K2 connect Administration console or the Service Object Designer to set the Default Destination before you publish your Service Object.

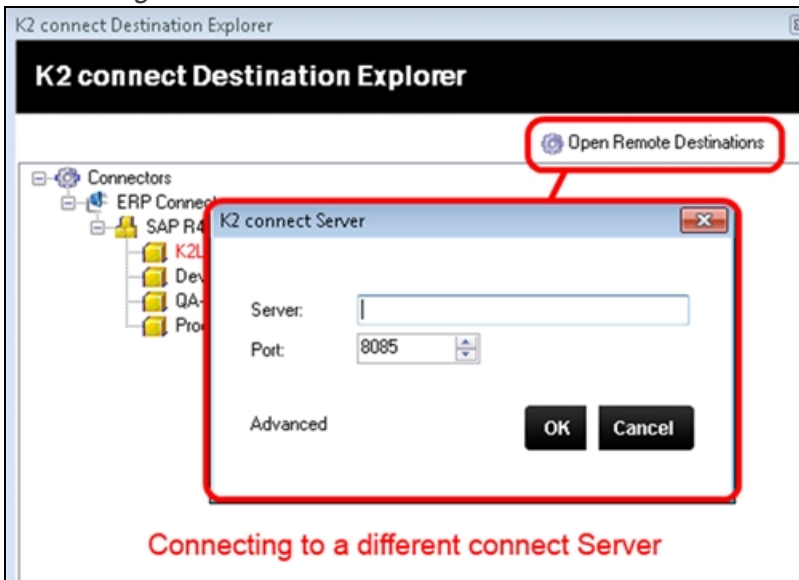
In the Destination Explorer, the current Default Destination will be marked in red. If this is not the Destination that you designed your connect Service Object against, use the right-click menu to set the appropriate Destination as the default Destination.

Setting the default destination



If you wanted to deploy your Service Object to a different K2 connect/blackpearl server, you will need to use the option to Open Remote Destinations and connect to the target K2 connect server, before selecting the Default Destination and publishing your Service Object.

Connecting to another K2 connect Server

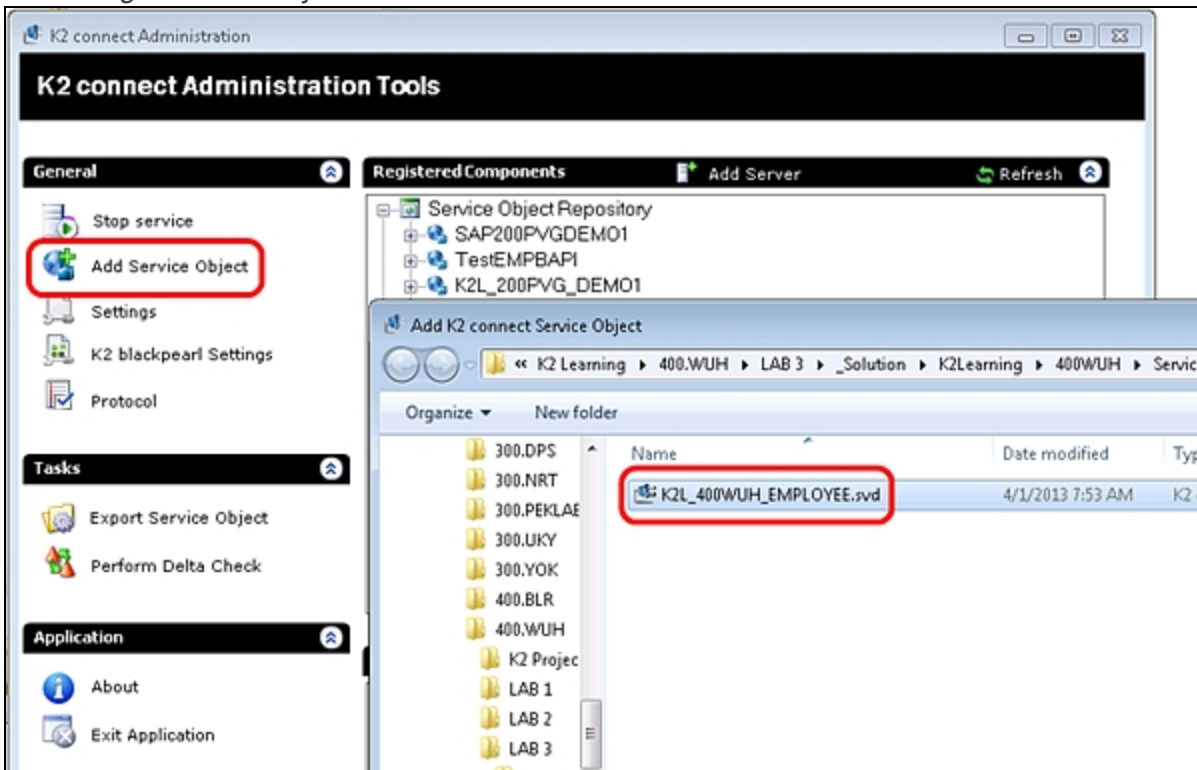


If you think back to the topic [What happens when you publish a Service object?](#), you may recall that the default connect Service Instance on the target K2 blackpearl server is refreshed as part of the publish process. If there are multiple connect Destinations on the target server, there will be multiple Service Instances as well, and you may need to refresh the Service Instance for the Destination you selected manually, since the Publish procedure only updates the first connect Service Instance.

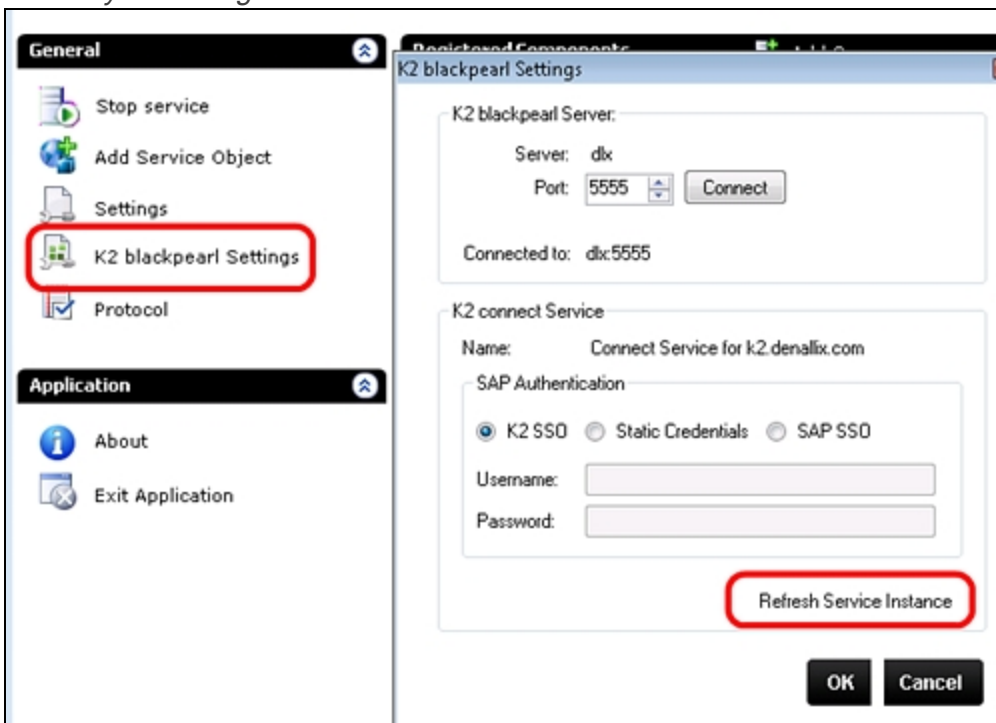
Now that the Service Instance is ready, you can deploy the associated K2 SmartObject to the target K2 blackpearl server, just like you would deploy any other SmartObject. Remember that the connect Service Object Publish procedure is complete separate from the K2 SmartObject deployment procedure, so these will usually be two separate steps. Also remember that, if there is an existing SmartObject on the target blackpearl server that uses the updated connect Service Object, you may need to restart the K2 blackpearl server as well, since the blackpearl server holds the connect proxy classes in memory as soon as the SmartObject is executed.

If you have an environment with strict change control policies, remember that you can also publish the connect Service Object using the connect Administration console. Just hand the .svd file to your connect Administrator and they can use the administration console to publish the .svd.

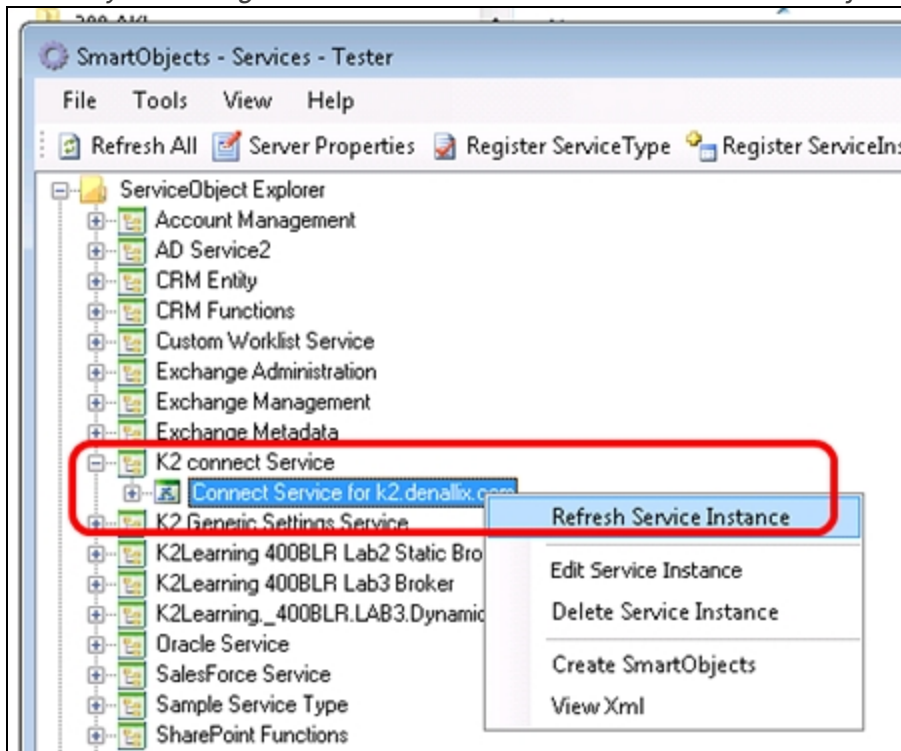
Publishing a Service Object with the K2 connect Administration console



Manually refreshing the default Service Instance with the connect Administration console



Manually refreshing the connect Service Instance with the SmartObject Service Tester utility



Review and Q&A

Review and Q&A

- Service Objects vs. SmartObjects
- Developer Tools
 - Service Object Explorer
 - Test Cockpit
 - SmartObject Tester Utility
- Creating Service Objects and K2 SmartObjects
 - Manually defining Service Methods
- Using Structures and XML for complex properties
- What happens when you publish connect Service Objects
- Destinations and the default destination
- Service Instances vs. Default Destinations
- Deploying to different K2 connect/blackpearl environments

Let's review the main topics covered in this learning module:

- Service Objects vs. SmartObjects
- Developer Tools
 - Service Object Explorer
 - Test Cockpit
 - SmartObject Tester Utility
- Creating Service Objects and K2 SmartObjects
 - Manually defining Service Methods
- Using Structures and XML for complex properties
- What happens when you publish connect Service Objects
- Destinations and the default destination
- Service Instances vs. Default Destinations
- Deploying to different K2 connect/blackpearl environments

Now that you know a little more about developing connect Service Objects and SmartObjects, think about the BAPIs you may want to expose in your own organization. How would you structure the Service Object methods, and can you think of reasons you may want to create separate Service Objects vs. adding multiple methods to the same Service Objects? How would you structure your Visual Studio projects to separate Service Objects from SmartObjects? Can you think of any potential pitfalls with deploying your Service Objects or SmartObjects?

This is also your opportunity to ask questions that clarify any of the topics covered during this module. If you are unclear about a topic, please ask the instructor to explain or elaborate.

Additional Resources

The following table lists additional resources that supplement the information provided in this module:

Resource	Location and Notes
KB article: Troubleshooting Auto Compile Errors	http://help.k2.com/en/KB000673.aspx <i>Discusses some common reasons why compilation might fail, and how to troubleshoot compilation errors.</i>
Connect group on K2	http://community.k2.com/t5/forums/searchpage/tab/message?filter=labels&q=k2+connect

Community	<i>The K2 underground community group for K2 connect</i>
KB article: How to deploy Service Objects in different environments	http://help.k2.com/en/KB000345.aspx <i>KB article that describes how to deploy connect Service Objects to different environments.</i>
Help guide: how to use transformation mappings to work with Binary data in SAP	<i>Development & User Guide > Filtering > Adding & Removing Filters > Data Mapping > Using K2 Connect Transformation mapping to return SAP binary data to SmartObjects</i> <i>Describes how to use custom transformation mapping code to convert a SAP binary property into a SmartObject base-64 binary property.</i>

Terms Of Use

ACCEPTANCE OF TERMS

THIS DOCUMENTATION IS SUBJECT TO THE FOLLOWING TERMS OF USE ("TOU"). SOURCECODE TECHNOLOGY HOLDINGS INC. ("SOURCECODE") RESERVES THE RIGHT TO UPDATE THE TOU AT ANY TIME WITHOUT NOTICE. THE MOST CURRENT VERSION OF THE TOU CAN BE REVIEWED BY CLICKING ON THE "TERMS OF USE" LINK IN THE TABLE OF CONTENTS.

COPYRIGHT

© 2008-2016 SOURCECODE TECHNOLOGY HOLDINGS, INC. ALL RIGHTS RESERVED. SOURCECODE SOFTWARE PRODUCTS ARE PROTECTED BY ONE OR MORE U.S. PATENTS. OTHER PATENTS PENDING. SOURCECODE, K2, K2 BLACKPEARL, K2 SMARTFORMS AND APPIT ARE REGISTERED TRADEMARKS OR TRADEMARKS OF SOURCECODE TECHNOLOGY HOLDINGS, INC. IN THE UNITED STATES AND/OR OTHER COUNTRIES.

THE NAMES OF ACTUAL COMPANIES AND PRODUCTS MENTIONED HEREIN MAY BE TRADEMARKS OF THEIR RESPECTIVE OWNERS.

THIS CONTENT AND ASSOCIATED RESOURCES ARE THE SOLE PROPERTY OF SOURCECODE TECHNOLOGY HOLDINGS, INC. AND SOURCECODE RESERVES ALL RIGHTS RELATED TO THE INFORMATION CONTAINED HEREIN. WITHOUT LIMITING RIGHTS UNDER COPYRIGHT, NO PART OF THIS CONTENT OR ITS RESOURCES, MAY BE REPRODUCED, STORED OR INTRODUCED INTO A RETRIEVAL SYSTEM, PRESENTED BY INDIVIDUALS OR ORGANIZATIONS IN TRAINING EVENTS OTHER THAN THOSE SANCTIONED BY SOURCECODE, OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, WITHOUT THE PRIOR WRITTEN CONSENT OF SOURCECODE TECHNOLOGY HOLDINGS, INC.

THIRD-PARTY INTEGRATION

SOURCECODE MAY MODIFY, ADJUST OR REMOVE FUNCTIONALITY IN THE K2 SOFTWARE IN RESPONSE TO CHANGES MADE TO VERSIONS OF THE RELEVANT MICROSOFT OR OTHER THIRD PARTY PRODUCTS. THIS DOCUMENTATION MAY PROVIDE ACCESS TO OR INFORMATION ON CONTENT, PRODUCTS, AND SERVICES FROM THIRD PARTIES. SOURCECODE TECHNOLOGY HOLDINGS INC. IS NOT RESPONSIBLE FOR AND EXPRESSLY DISCLAIMS ALL WARRANTIES OF ANY KIND WITH RESPECT TO THIRD-PARTY CONTENT, PRODUCTS, AND SERVICES.

NOTICE REGARDING CONTENT OF THIS DOCUMENTATION

THE INFORMATION CONTAINED IN THIS DOCUMENT AND ITS ASSOCIATED RESOURCES, INCLUDING UNIFORM RESOURCE LOCATORS AND IDENTIFIERS, IS SUBJECT TO CHANGE WITHOUT NOTICE.

UNLESS EXPLICITLY STATED OTHERWISE, THE PEOPLE, ORGANIZATIONS, COMPANIES, PLACES, DOMAIN NAMES, E-MAIL ADDRESSES, PRODUCTS, LOGOS, AND EVENTS DEPICTED ARE FICTITIOUS AND NO ASSOCIATION WITH ANY ACTUAL PEOPLE, ORGANIZATIONS, COMPANIES, PLACES, DOMAIN NAMES, E-MAIL ADDRESSES, PRODUCTS, LOGOS, AND EVENTS IS INTENDED, OR SHOULD BE INFERRED UNDER ANY CIRCUMSTANCE.

WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS DOCUMENTATION, THE DOCUMENTS AND RELATED GRAPHICS CONTAINED IN THIS DOCUMENTATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. SOURCECODE TECHNOLOGY HOLDINGS INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE DOCUMENTATION AT ANY TIME AND WITHOUT NOTICE. ERRORS AND INACCURACIES MAY BE REPORTED IN WRITING TO DOCUMENTATION@K2.COM