Information Management software

Redpaper

Paolo Bruni
Michael Schenker

IBM Data Studio V2.1: Getting Started with Web Services on DB2 for z/OS

Introduction

This IBM® Redpaper™ provides a step-by-step introduction on how to get started with Web services using Data Studio V2.1. Our goal is to explain basic concepts and introduce the functions that can help you take your first steps in the Web services arena with DB2® for z/OS®. We start with describing and setting up the basic software components and showing a step-by-step guide to creating your first Web service. We then show how you can test and deploy Web services and let DB2 operate as a Web service requester (consumer) and a Web service server (provider). Last, we introduce the SOAP user-defined functions (UDFs) that enable DB2 to act directly as a Web service consumer, and we show how stylesheets can easily transform your Web service response into a service message in HTML format.

The SQL examples that we use are extremely simple. For detailed implementation of more complex business cases, refer to *DB2 9 for z/OS: Deploying SOA Solutions*. SG24-7663.

This paper contains the following topics:

- Required software components
- Generating and deploying a Web service
- Testing your Web service
- Invoking the Web service with SOAP UDFs
- What stylesheets can do for you
- Hints and tips for Data Web Services
- Enabling stored procedures for JDBC support

Disclaimer

This paper is current as of March 2009 and contains procedures and screen shots relevant to Data Studio Developer V2.1. Your version might differ and show dissimilarities.

Required software components

We use two workstation components: IBM Data Studio Developer, which includes functions to create Web services and IBM WebSphere® Application Server Community Edition, which provides functions to deploy the Web services. You can download both components from the Web. Having DB2 for z/OS installed enables you to step through this Web services primer without having to purchase any additional IBM software. We highlight the steps in this document that require DB2 9 for z/OS or might require a separate function.

IBM Data Studio

Data Studio is the strategic IBM tool that is meant to increase productivity and reduce development costs throughout the data life cycle by providing an Integrated Data Management Environment. Individual plug-in tools provide powerful capabilities that target specific data management roles, but the components interoperate seamlessly, which enables cross-role collaboration, productivity, and effectiveness. The integration extends into the Rational®, WebSphere, and Tivoli® portfolios. Data Studio consists of:

- Rational Data Architect

  The data architect's key tool is Rational Data Architect, which is the tool for discovering, modeling, relating, and standardizing data. It supports logical and physical modeling and automation features for diverse databases that simplify tasks, such as reverse engineering from existing databases, generating physical models from logical models, generating Data Definition Language (DDL) from physical models, and visualizing the impact of changes.

- IBM Data Studio Developer

  Data Studio Developer is an Integrated Development Environment for creating and testing database objects, queries, database logic, and pureQuery applications. It:

  - Develops database applications faster with an integrated query editor for SQL and XQuery
  - Optimizes your applications and queries with ease
  - Builds and tests your stored procedures (Java™ and SQL) with the interactive routine debugger
  - Rapidly develops or customizes the SQL inside Java applications
  - Auto-generates test applications for your pureQuery applications
  - Provides heterogeneous data server support for IBM DB2 Data Server and Informix® Dynamic Server across the platforms

- IBM Data Studio pureQuery Runtime

  Data Studio pureQuery Runtime is a high-performance Java data access platform. It improves security and manageability of Java application connections to databases. It is an innovative approach to building high quality, better performing database applications while greatly improving Java programmer productivity. It:

  - Bridges the gap between data and Java technology by harnessing the power of SQL within an easy-to-use Java data access platform
  - Improves the security and performance characteristics of your Java applications
  - Deploys applications to query in-memory collections and databases using a single API
  - Deploys advanced pureQuery applications using static SQL to improve application performance and management to deliver predictable responses

- IBM Data Studio Administrator

  You can use IBM Data Studio Administrator to increase DBA productivity and to reduce application outages by automating and simplifying complex DB2 structural changes.

IBM Data Studio currently supports developing code for IBM DB2 and Informix Dynamic Servers (IDS), more specifically:

- DB2 for Linux®, UNIX®, Windows® v8.x, v9.1.*x*, and v9.5
- DB2 for z/OS V8 and DB2 9 for z/OS
- DB2 for i5/OS® V5R2, V5R3, and V5R4
- Informix Dynamic Server (IDS) v9.*x*, v10.*x*, and v11

For more information, refer to:

http://www.ibm.com/software/data/studio/

Data Web Services

Data Web Services (DWS) is the set of functions included in the Data Studio Developer component that provides the capability to generate Web services-based access to your database. DWS is the solution to significantly ease the development, deployment, and management of Web services-based access to DB2 and IDS database servers. DWS is mainly a bottom-up approach to expose existing functionality (stored procedures or SQL statements) as Web service operations:

- Using DWS, you can take Data Manipulation Language (DML) statements, such as Select, Insert, Update, Delete, and XQuery, and stored procedures and generate Web services without writing a single line of code.
- DWS provides a full Web-service interface, which includes support for SOAP and REST-styled bindings.

These features are part of Data Studio Developer, which means that you can develop Web services and database applications in one environment. The generated Web services are packaged in the form of a ready-to-deploy Web application, which you can then deploy to supported application servers.

The key aspects of DWS are:

- Creating Web services using DWS requires no programming:
  - DWS lets you create Web services using a drag-and-drop interface: Drag and drop any Data Manipulation Language (DML) operation or stored procedure into a Web service container to create ready-to-deploy Web services.
  - DWS also supports an integrated test environment that lets you deploy and test the generated services using a few clicks of the mouse.
- DWS supports SOAP over HTTP and Web Services Description Language (WSDL) generation:
  - DWS automatically generates a WSDL file that contains a description of the Web services.
- DWS supports the REST-style service interface:
  - In addition to SOAP over HTTP, DWS supports the HTTP GET/POST binding for provisioning REST-styled services to your database server.

- DWS can apply server-side Extensible Stylesheet Language Transformation (XSLT) to incoming and outgoing XML service requests and responses:
  - DWS lets you apply server-side XSLT to match any service format requirements that you might have. This approach has interesting possibilities in the Web 2.0 world.
- No code generation:
  - DWS consists of a common metadata-driven runtime, and there is no black box code that gets generated under the covers, which results in a reliable and lightweight application.

IBM WebSphere Application Server Community Edition

IBM WebSphere Application Server Community Edition, a member of the WebSphere product family, is a lightweight Java Platform, Enterprise Edition 5 (JEE5) application server that is built on Apache Geronimo (the open source application server project of The Apache Software Foundation).

IBM WebSphere Application Server Community Edition is designed to help accelerate your development and deployment efforts by harnessing the latest innovations from the open source community and to help provide a readily accessible and flexible foundation for building Java applications.

For more information about current WebSphere Application Server Community Edition versions, functions, and IBM support, visit:

http://ibm.com/websphere/wasce

WebSphere Application Server Community Edition includes the baseline components that you need to deploy your applications in one integrated package. Pre-integration with the Apache Derby database helps to deliver a full-featured, robust, small-footprint database server that you can use during development and deployment stages.

You can remove unnecessary components to further streamline the footprint, or you can extend the capabilities of the server with custom features.

The key product features are:

- Support for Java Platform, Standard Edition (Java SE), Version 5.0 makes it possible for your WebSphere Application Server Community Edition applications to take advantage of the many language innovations in this version, including annotations, enumerated types, generics, and improved performance. You can also now use components from other vendors that require Java SE, Version 5 as part of your applications.
- Java EE 5 focuses on making development easier but retains the richness of the J2EE™ 1.4 platform that established Java EE as the premier platform for Web services and enterprise application development.
- Apache Geronimo brings together best-of-breed technologies across the broader open source community to support Java EE 5 specifications.
- The Eclipse plug-in for WebSphere Application Server Community Edition helps you take advantage of the Eclipse technology-based Web Tools Platform, Version 2.0. This plug-in offers a simple development environment for creating and debugging your WebSphere Application Server Community Edition applications. Using the WebSphere Application Server Community Edition JEE profile rules for Eclipse Test and Performance Tools Platform (TPTP), you can help ensure that the applications that you create are portable and deliver increased flexibility.

- Web-tier clustering helps improve reliability for your applications by providing failover support. Using this feature, you can also scale (up or down) and customize according to your business needs. You can achieve load balancing when you use WebSphere Application Server Community Edition in conjunction with Apache HTTP Server and the mod-jk connector or mod-proxy support for Apache JServe Protocol (AJP).
- Flexible deployment options and an integrated development environment enable you to rapidly develop and deploy multiple configurations, whether as stand-alone servers or as endpoints in a WebSphere service-oriented architecture (SOA) infrastructure. Version 2.0 adds the ability to redeploy existing deployed applications from the console and the command line, hot deployment for compressed Java Archive (JAR) files, and in-place deployment for exploded JEE modules.
- The embedded, open source, Apache Derby database delivers a robust, small-footprint database server with full transactional capability that is simple to deploy and helps reduce the cost of Web-based and embedded applications. If you want external database access, WebSphere Application Server Community Edition also provides driver support for IBM DB2 Universal Database™, Oracle®, Microsoft® SQL Server®, and MySQL™.
- You can tailor WebSphere Application Server Community Edition to your needs and help improve performance while helping to save system resources by selectively enabling and disabling specific components of WebSphere Application Server Community Edition at run time.
- WebSphere Application Server Community Edition features centralized user management by unifying all applications that run on the platform under a shared, standards-based security framework that is based on Java Authorization Contract for Containers (JACC). This capability virtually eliminates the complexity of setting up and maintaining separate user-management systems for your applications. WebSphere Application Server Community Edition also supports Lightweight Directory Access Protocol (LDAP) authentication.

  WebSphere Application Server Community Edition delivers an easy-to-use administrative console application that lets you manage and monitor the application server and related resources. The updated console in Version 2.0 is even easier to use and provides more valuable information for monitoring your server and your applications.

IBM Data Studio Developer

If you have never worked with Data Studio Developer before, and if you do not have a running instance of Data Studio Developer V2.1 on your workstation, you must:

1. Download the Data Server code from the Web.
2. Install it.

Downloading Data Studio Developer V2.1

To download Data Studio Developer:

1. Click the following hyperlink:

   `http://www.ibm.com/developerworks/downloads/im/datastudiodev/`

   Figure 1 on page 6 shows the Web site content that is displayed. The site gives you a chance to download no-charge trial code of IBM Data Studio Developer.

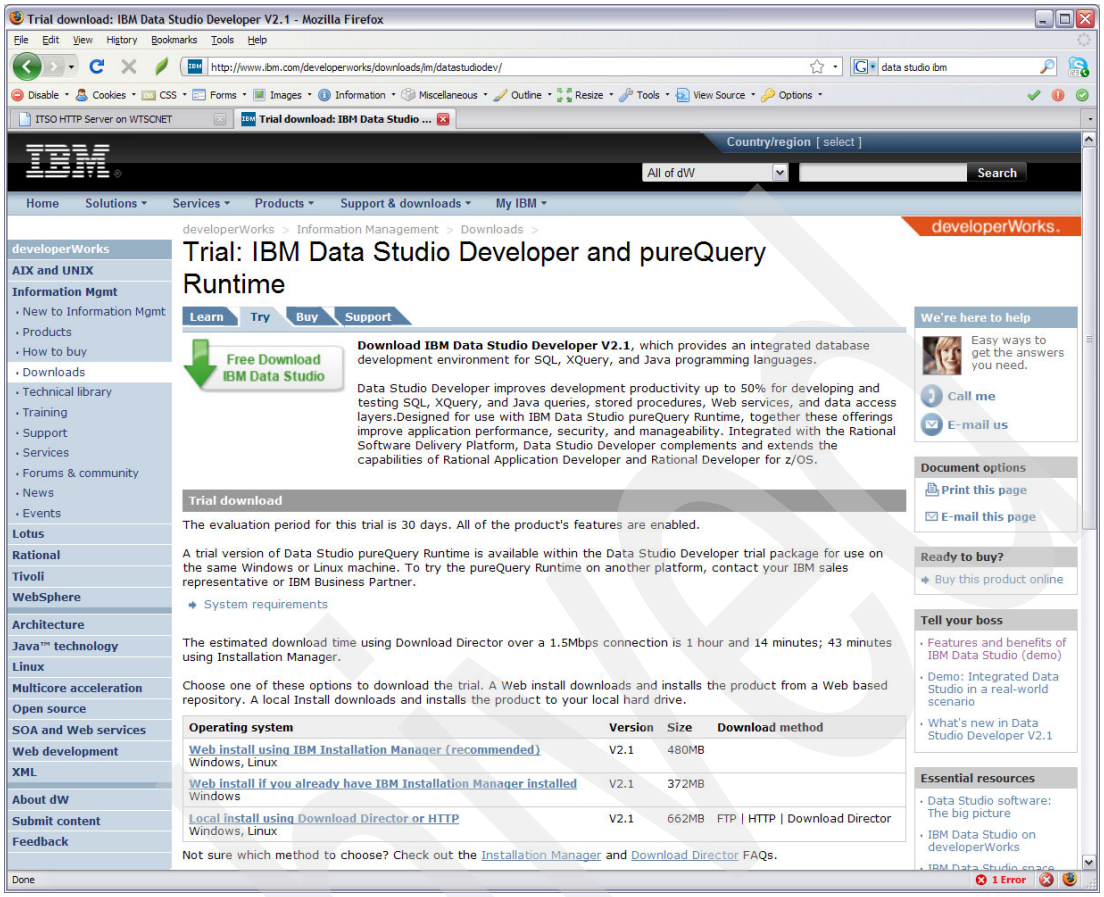


Figure 1 Data Studio Developer Download Web site

> **Note:** The Web site mentions 30-day trial code. After you finish the 30-day trial phase, you cannot use advanced features any more, unless you purchase a license.
>
> After the 30 days, you can just continue using the free functions of IBM Data Studio Developer.

2. IBM Data Studio Developer requires that you install DB2-provided Java Database Connectivity (JDBC™)/Open Database Connectivity (ODBC) stored procedures, which you install by executing member DSNTIJSG in your hlq.SDSNSAMP library during the DB2 installation process. If these stored procedures are not created at DB2 installation time, you need to customize and run job DSNTIJMS to define the stored procedures.

   In “Enabling stored procedures for JDBC support” on page 74, we show common issues related to these stored procedures after maintenance is applied to DB2 code, which changed any of the packages involved.

Installing IBM Data Studio Developer

The Web site provides you with three options on how to download and install IBM Data Studio Developer:

- Web install using IBM Installation Manager
- Web install if you already have IBM Installation Manager installed
- Local install using Download Director or HTTP

We choose the first option **Web install using IBM Installation Manager**:

1. Click the link to download IBMIM_win32.exe to your machine and execute it. You will be prompted with a password dialog where you provide your credentials for the IBM download site. Refer to Figure 2.

Figure 2 Password prompt for the IBM download site

2. A new dialog asks for the version that you want to install. As shown in Figure 3, IBM Data Studio Developer V2.1 is preselected. You might also see a later version of the IBM Installation Manager in the list. Click **Next** to continue the installation process.

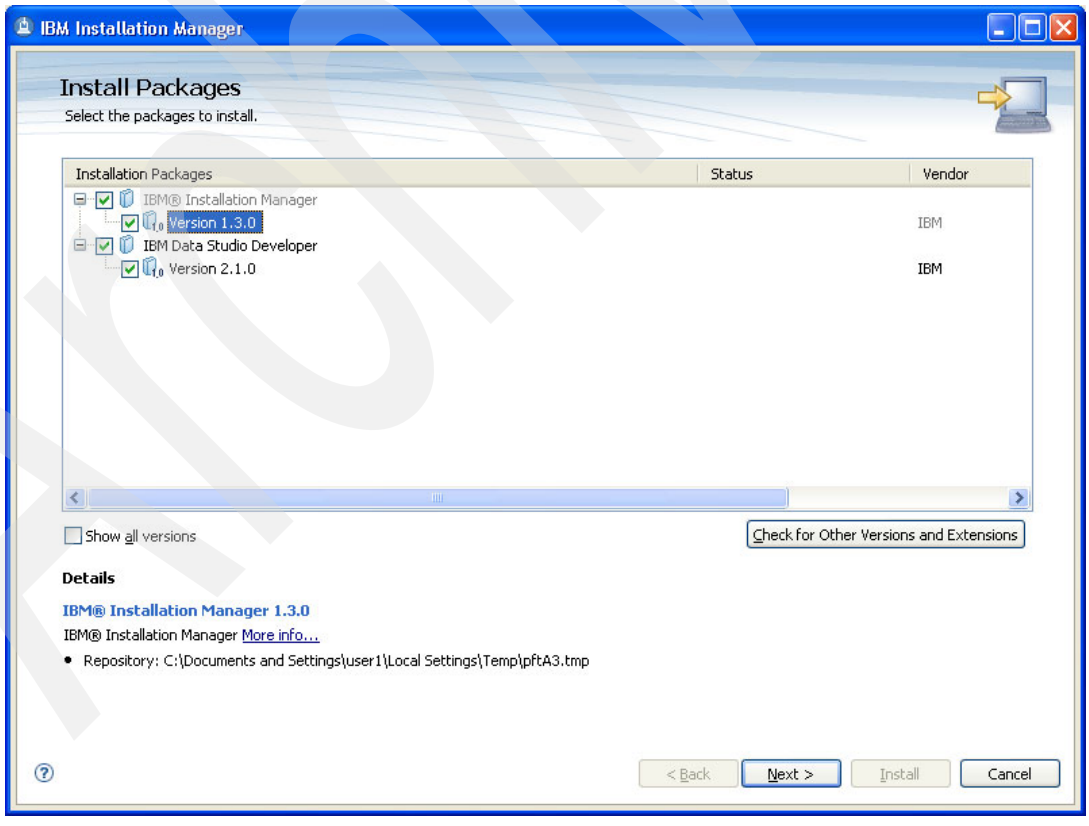


Figure 3 Install the package selection

3. The next window displays the license agreement. After agreeing to the license terms, click **Next**.

4. After answering a few questions about you and the usage of the product, the system prompts you to provide a location for the shared resources directory and the location for the Installation Manager. Click **Next** to accept the directory selection.
5. In the next dialog, you specify the installation directory for the IBM Data Studio package group. Click **Next** to accept the directory selection.
6. Click **Next** in the following dialog if you do not want to extend an existing Eclipse installation.
7. When you get to the window that is shown in Figure 4, you decide which languages you might want to use in the presentation logic of Data Studio. Use this page to select the languages to install for this package. The corresponding language translations for the user interface and the related documentation are installed. If you are a travelling user, it might be a good idea to install at least English in addition to your home language. In our installation, we decided to select German and French in addition to English.

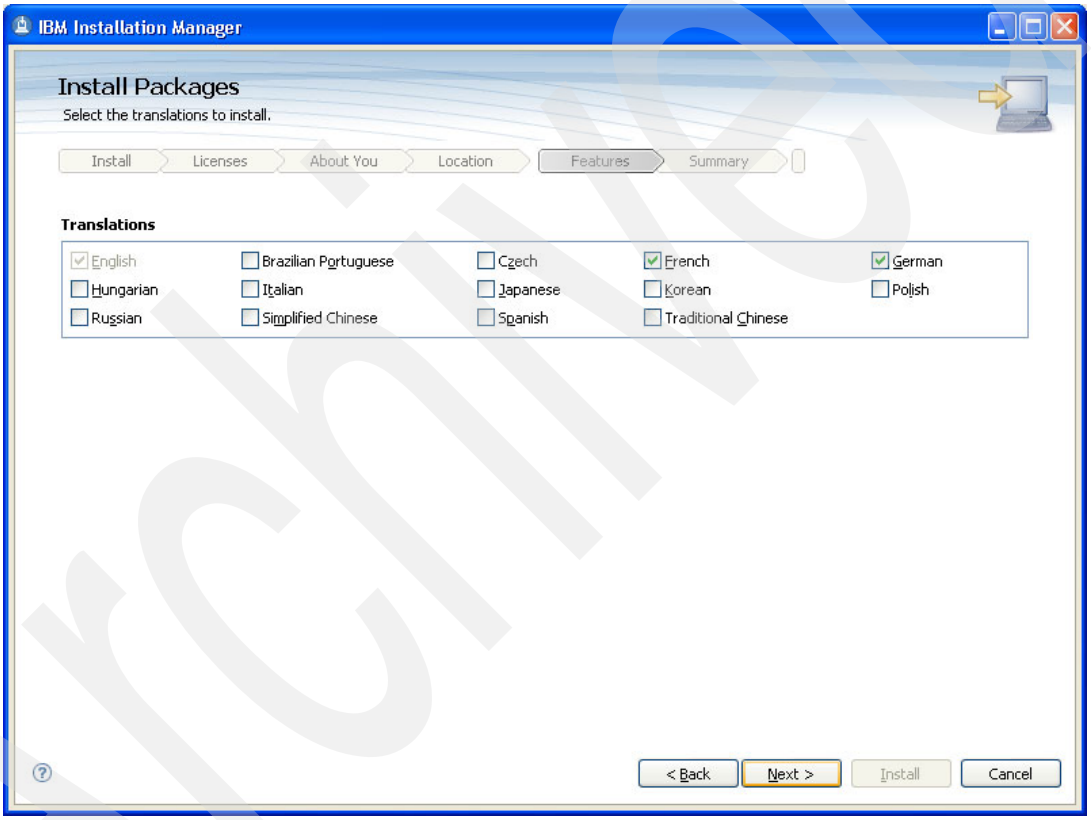


Figure 4 Install language package

8. Depending on the functions that you want to use with your new IBM Data Studio Developer tool in the future, you can select or clear the offered components of IBM Data Studio Developer. Refer to Figure 5 on page 9. Because we are satisfied with the preselected components, we make no changes. In case you want to install those components more selectively, the space requirements that are reported at the bottom of Figure 5 on page 9 will change in accordance with your selections.

Note: All functions listed under *Advanced Features* require a license to use them permanently. After the trial period of 30 days, you have to uninstall those features if no license is available, otherwise Data Studio cannot be used. But you can just continue using the no-charge functions of IBM Data Studio Developer.

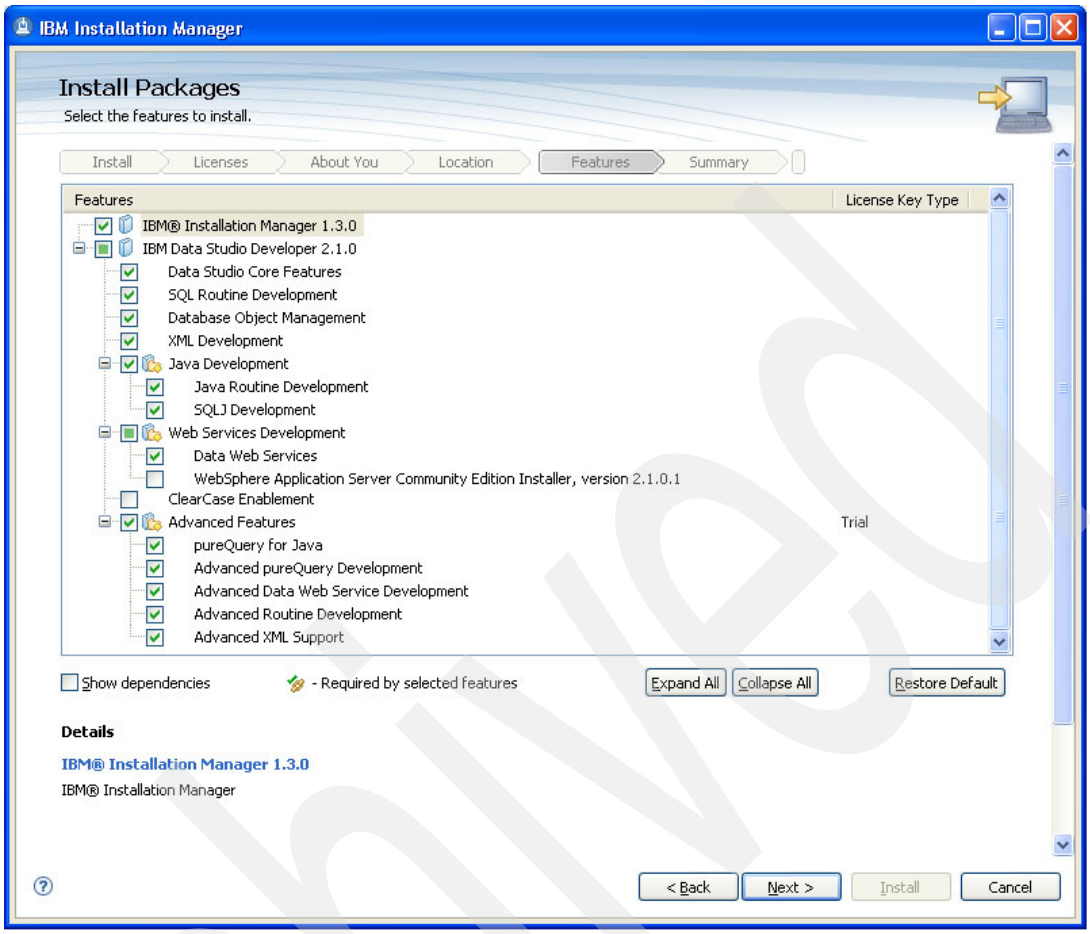


Figure 5 Install components and space requirements

9. Click **Next**. A summarized report about what is to be installed on your machine is displayed.
10. Click **Finish** to begin the installation.

Getting WebSphere Application Server Community Edition

After you create a Web service using Data Studio tooling, you must deploy your newly created Web service in an environment from which it can be invoked. One no-charge possibility for doing so is to use WebSphere Application Server Community Edition. WebSphere Application Server Community Edition is an open source, lightweight Java EE application server that provides a readily accessible and flexible foundation for building Java applications. We have introduced WebSphere Application Server Community Edition in “IBM WebSphere Application Server Community Edition” on page 4.

The current version of WebSphere Application Server Community Edition is V2.1. Refer to:

`http://www.ibm.com/software/webservers/appserv/community/`

We used Version 2.1.1.1 for the examples in this Redpaper. You can download the version from:

`http://www.ibm.com/developerworks/downloads/ws/wasce/`

After the download completes, execute **wasce_setup-2.1.1.1-win.exe** on your workstation. The installation completes successfully. You can verify the success of the installation with a startup of the WebSphere Application Server Community Edition server by selecting **All Programs → IBM WebSphere → Application Server Community Edition → Start the server**.

A Java runtime window starts, and after approximately one minute, the server starts successfully giving the information that is shown in Figure 6.

Figure 6 WebSphere Application Server Community Edition startup message in Java window

Now, you have the prerequisite workstation software, and you are ready to create your first data Web service.

Generating and deploying a Web service

In this section, we describe how you can get started with the IBM Data Studio Developer tooling for Web services. We show how you can easily build a Web service, the required actions to deploy the Web service to a J2EE-compliant server (WebSphere Application Server Community Edition in this case), and how you can start your new Web services from different platforms.

After we show the deployment of the Web service, in the following sections, we also introduce the DB2 for z/OS consumer UDFs and explain how you can start using them. To explain the whole picture, we conclude by showing how pureXML® can help when you use Web services.

Setting up Data Studio

We mentioned IBM Data Studio Developer as a required component for this tutorial scenario earlier in this Redpaper. We assume that Data Studio is installed on your workstation using the simple instructions in “IBM Data Studio Developer” on page 5.

Starting Data Studio

To start Data Studio Developer on your workstation:

1. Select **Start** → **All Programs** → **IBM Data Studio** → **Data Studio Developer**, and start the client program.
2. If you have never used Data Studio Developer before, you first see a Welcome window. For our purposes, we just close the Welcome window. Next, the panel in Figure 7 is displayed.

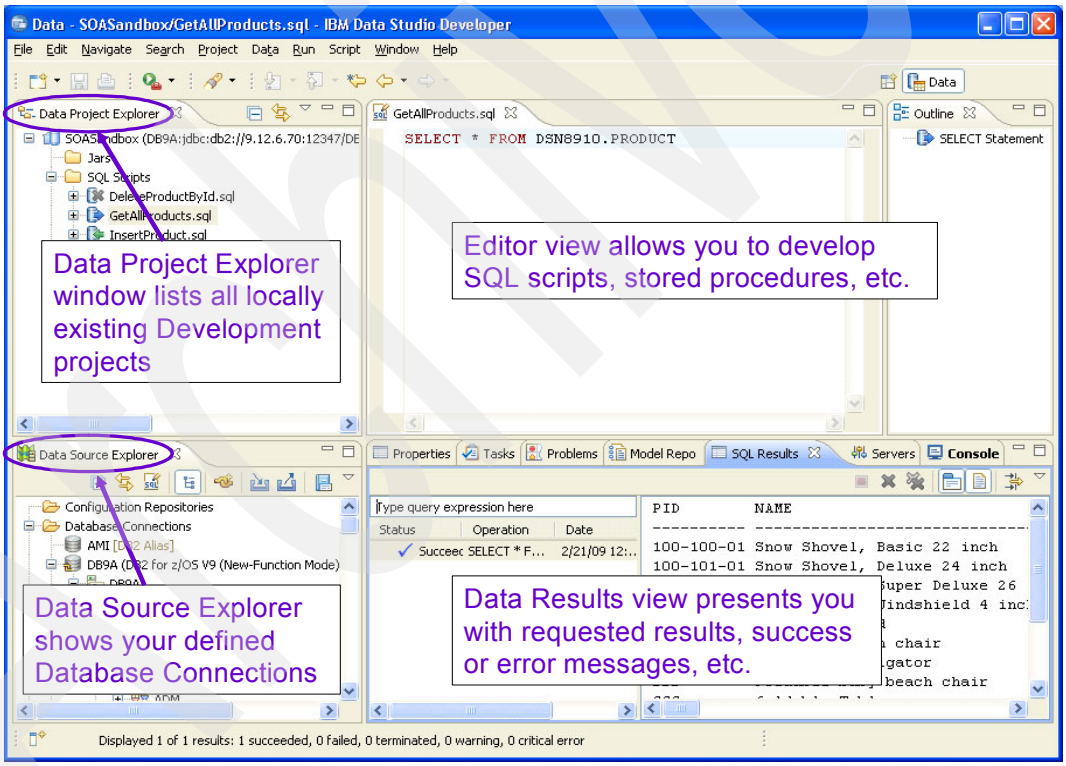


Figure 7 Data Studio first window

It is important to understand the views that make up the full window of your Data Studio GUI. The four views of Data Studio, illustrated in Figure 7, are:

- Data Source Explorer

  In the Data Source Explorer, you can connect to existing databases and view their designs and objects. Those DB2 databases can either exist locally on your workstation or be on other platforms, such as AIX® or z/OS.

Using the Data Source Explorer, you can:

- Create and manage database connections and browse data objects in a connection
- Define connection filters
- Connect to existing databases and view their designs
- Reconnect to a database, if your database connection was disconnected
- Disconnect from a database connection, if you are connected
- Delete a database connection
- Import or export database connection information to an XML file
- Modify data objects, and manage changes
- Debug stored procedures using the integrated debugger

- Data Project Explorer window

  In the Data Project Explorer, you can work locally with data objects.

  Data development projects are used for database application development. This type of project is associated with a single connection in the Data Source Explorer. Use data development projects to develop the following resources:

  - You can develop, test, and deploy SQL and Java stored procedures and UDFs. If you want to learn more about these functions as provided by Data Studio, refer to Chapter 27, “The IBM Data Studio”, and Chapter 28, “Tools for debugging DB2 stored procedures”, in *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond,* SG24-7604.
  - If the target server supports XML, you can develop XML files and artifacts for XML applications.
  - You can develop and test SQL queries.
  - You can develop and deploy Web services that access data by using SQL scripts or stored procedures. In “Preparing for your first Web service” on page 17, we focus on this functionality.

- Data Results view

  The Data Results view displays the output and status of certain actions against database objects and data definitions, such as the status and resulting output of a run action against a stored procedure. The Data Results view maintains the status and output history of each unique action and object or definition pair.

- Editor view

  The Editor view provides editing options for SQL statements, scripts, stored procedures, XML files, and so forth. Several editors provide advanced features, such as syntax highlighting and auto-completion.

Setting up a connection to a database server

Before you can start with the development of a few SQL scripts and one very simple stored procedure, which later we combine in a first extremely simple Web service, you must define a connection to a database server. Unless stated differently, we associate all of the following steps in this section with a DB2 9 for z/OS subsystem. We also assume that the distributed data facility (DDF) address space is set up correctly and it is ready to use; however, equivalent functions are available when connecting to a DB2 for Linux, UNIX, and Windows database server.

To set up a connection to a database server:

1. Issue the DB2 command -DIS DDF for your DB2 9 for z/OS subsystem. Figure 8 on page 13 shows the result.

```
RESPONSE=SC63
 DSNL080I  -DB9A DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
 DSNL081I STATUS=STARTD
 DSNL082I LOCATION           LUNAME            GENERICLU
 DSNL083I DB9A               USIBMSC.SCPDB9A   -NONE
 DSNL084I TCPPORT=12347 SECPORT=12349 RESPORT=12348 IPNAME=-NONE
 DSNL085I IPADDR=::9.12.6.70
 DSNL086I SQL    DOMAIN=wtsc63.itso.ibm.com
 DSNL086I RESYNC DOMAIN=wtsc63.itso.ibm.com
 DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

Figure 8 -DIS DDF command output

You now know the IP address, the port number, and the location name of the DB2 subsystem, and we can define the database connection from within Data Studio.

2. In the Data Source Explorer view, right-click **Connections**.
3. Select **New Connection**, and then click **DB2 for z/OS - All Versions**. The window in Figure 9 is displayed.

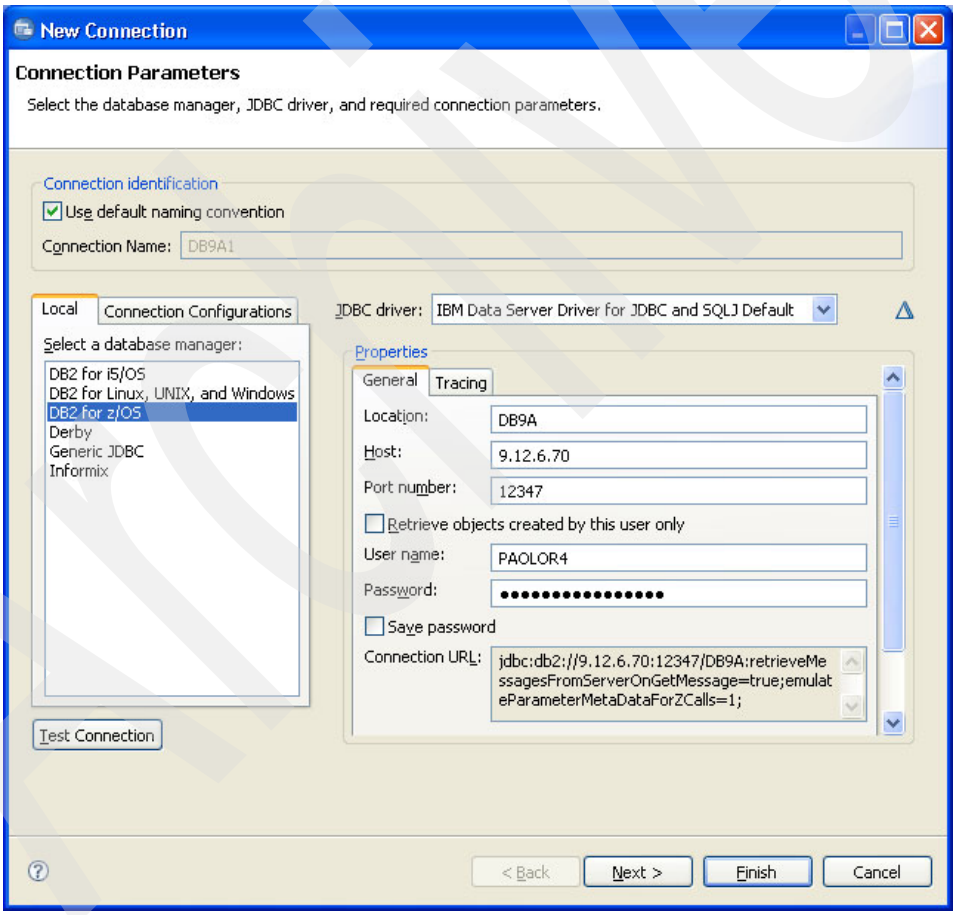


Figure 9 Connection parameters

4. Populate the empty fields using the following data:
   - Connection name: `DB9A`
   - Location: `DB9A`

Important: You must enter the location name in uppercase letters.

- Host: 9.12.6.70
- Port: 12347

5. On the left side of the window, enter the user information. The User name and Password fields are not case-sensitive.
6. Click **Test Connection** to check that the definition is correct.
7. If your connection definition is correct, you can connect to the DB2 for z/OS subsystem. Click the newly created definition on the Data Source Explorer, and select **Reconnect**. Enter your user credentials, and wait until you are connected. If the connection works fine, the icon to the left of your connection name turns green, and a + (plus) sign is displayed to the left of the icon as well, as shown for connection DB9A in Figure 10.

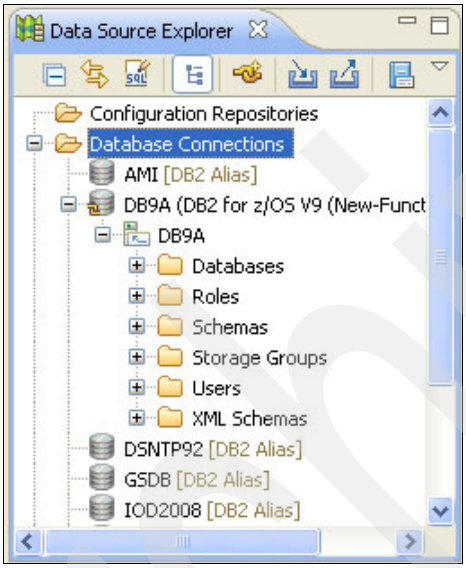


Figure 10 Active database connections

Creating a new Data Development Project

After you have established the database connection, the next step is to create a Data Development Project. Although the Data Development Projects are locally stored on the workstation, they are always tied to a specific database connection, one at a time.

To create a new Data Development Project:

1. Right-click anywhere in the Development Project view. A new window is displayed, where you select **NEW** → **Data Development Project** as shown in Figure 11 on page 15.

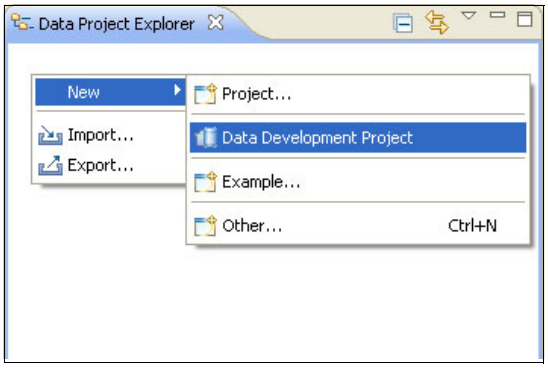


Figure 11 Creating a Data Development Project

2. On the next window, you specify the project name. In Figure 12, we enter `SOASandbox` as the project name in this case.

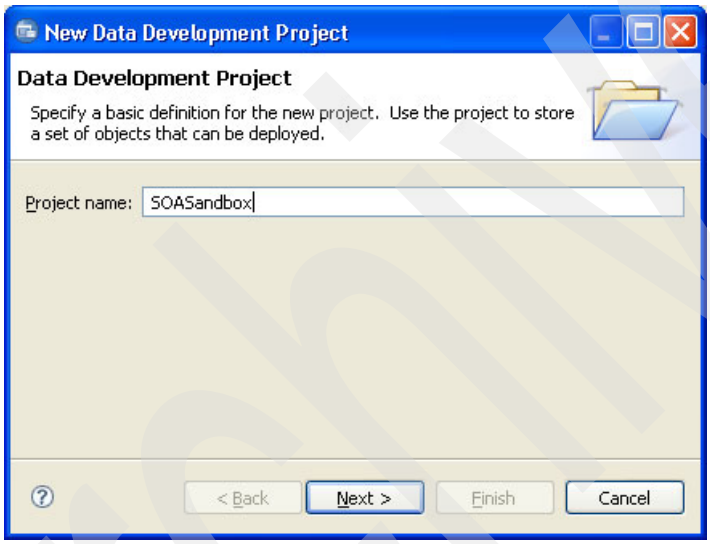


Figure 12 Data Development Project name

3. Click **Next**. Each project is tied to a specific database connection. In our case, we choose to use the existing database connection of “Setting up a connection to a database server” on page 12, which is DB9A, as shown in Figure 13 on page 16.

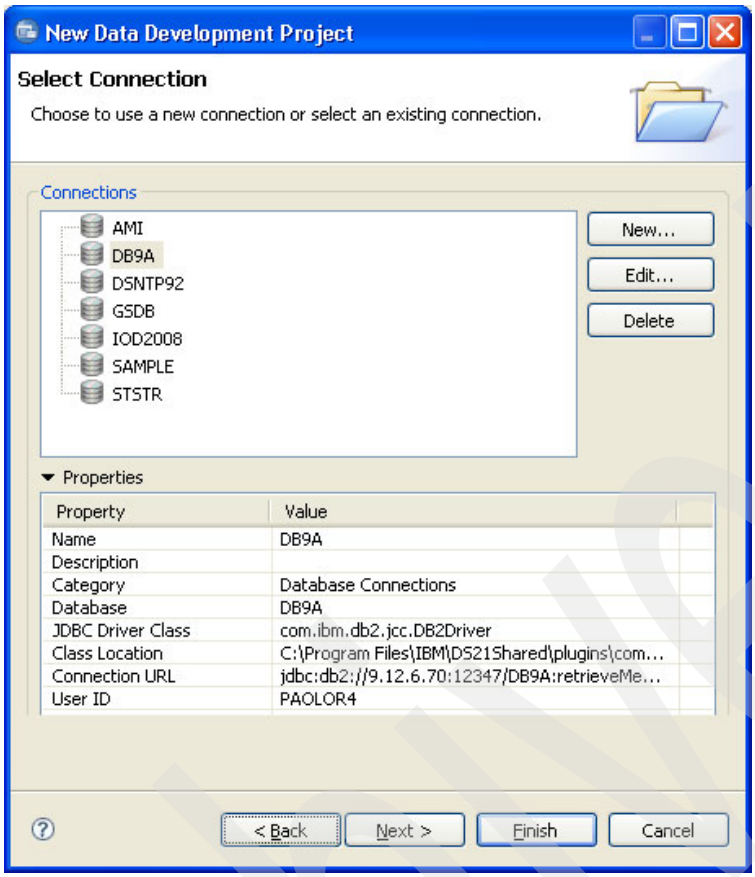


Figure 13 Data Development Project: Select Connection

If we had not already set up the database connection, we use Create a new connection here, and the wizard takes us to the same window of Figure 9 on page 13.

4. Click **Next**. In the next dialog, you can provide information about the package and the build owner. This information is not needed for this example. Simply click **Next**.
5. In Figure 14 on page 17, you can now select the authorization ID to use as the current schema for the project and whether the system will omit the current schema in generated SQL statements.

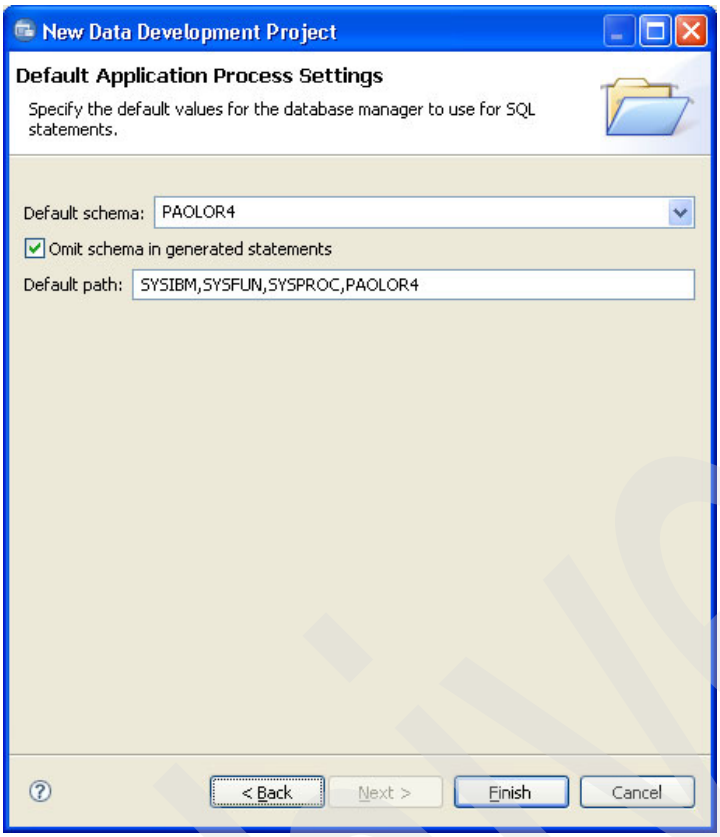


Figure 14 Select default schema and default path

6. Click **Finish**, and the project is created.

Preparing for your first Web service

For the first Web service, we need to define a couple of database accesses. To show that you can equally and easily implement all four major DML types, we create four SQL scripts: one each for SELECT, INSERT, UPDATE, and DELETE. To keep it as simple as possible for the purpose of demonstrating the pure functionality of creating and using Web services, we only reference a single table in each one of the four SQL scripts.

Notice how simple it is to include existing stored procedures within the Web service. We include a simple native SQL stored procedure that accesses the DB2 catalog. We show you how to quickly create this native stored procedure for later use.

Creating SQL scripts

The table definition for table DSN8910.PRODUCT is shown in Figure 15 on page 18. This table is one of the sample tables that the installation verification jobs of DB2 9 for z/OS have created.

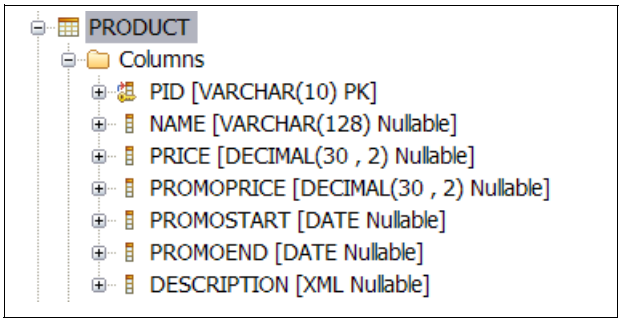


Figure 15 DSN8910.PRODUCT table definition

If the table does not yet exist on your DB2 subsystem or if you are on DB2 V8, you can use the CREATE TABLE statement that is shown in Example 1. If you are working with DB2 V8, remember to omit the column DESCRIPTION, which is defined as the new XML data type in DB2 9.

Example 1 CREATE DSN8910.PRODUCT table DDL

```
CREATE TABLE DSN8910.PRODUCT
  ( PID              VARCHAR(10)   NOT NULL PRIMARY KEY
   ,NAME             VARCHAR(128)
   ,PRICE            DECIMAL(30, 2)
   ,PROMOPRICE       DECIMAL(30, 2)
   ,PROMOSTART       DATE
   ,PROMOEND         DATE
   ,DESCRIPTION      XML )
```

We now show how to create the four SQL scripts in the Data Studio Data Development Project.

Creating SQL script InsertProduct

To create the SQL script InsertProduct:

1. Open the project.
2. Right-click **SQL Scripts**.
3. Select **New** → **SQL or XQuery Script**. A new window is displayed, as shown in Figure 16 on page 19.

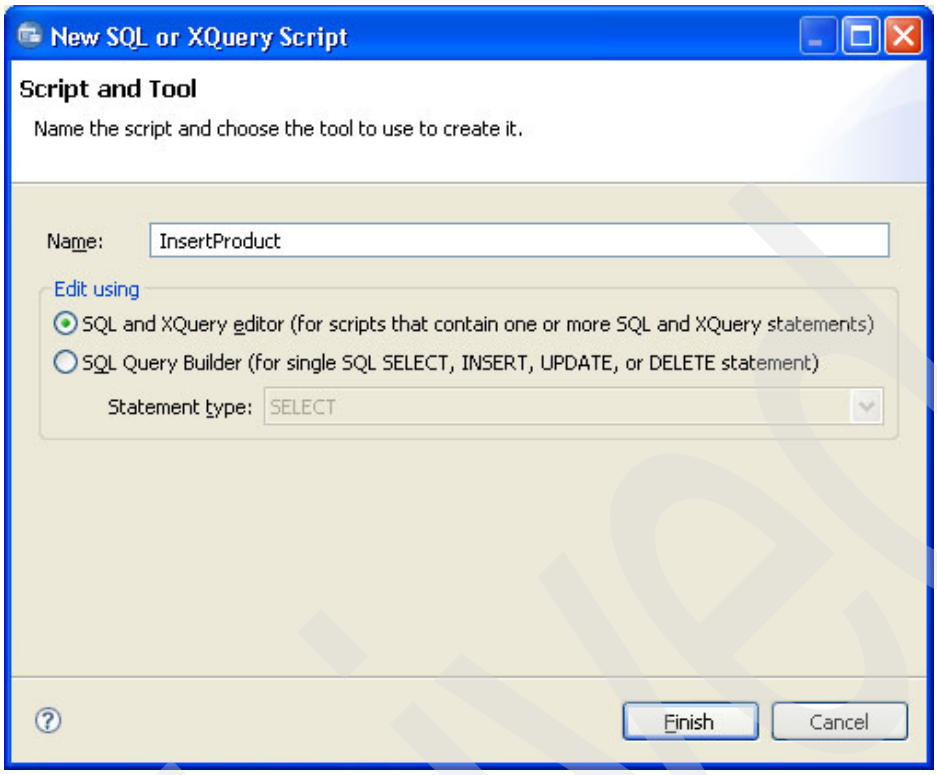


Figure 16 Create new SQL script

4. Select the project that you have just created, as described in "Creating a new Data Development Project" on page 14:
   a. Change the default name of the statement from `Script1` to `InsertProduct`.
   b. Keep the preselected option **SQL and XQuery Editor.**

      On the upper right corner of your main window, the Editor view of Data Studio Developer is displayed for the first time, as shown in Figure 17 on page 20.

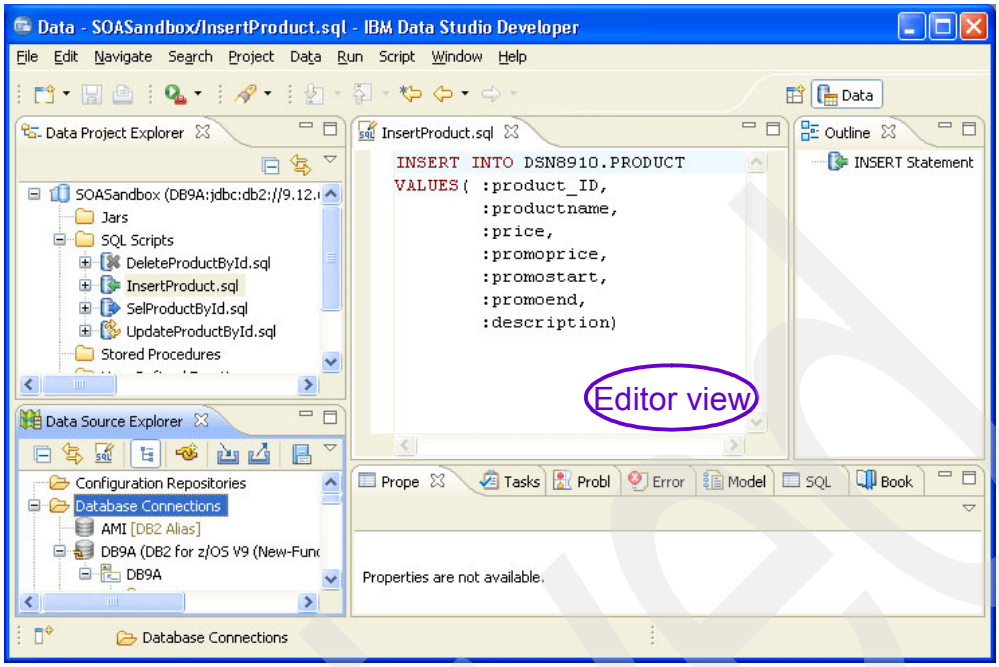


Figure 17 Edit SQL statement in Editor view

5. Click **Finish**.
6. Type the INSERT statement, as shown in Figure 17, and click in the Editor view. When you finish editing, you can click the little **x** in the Editor view next to the SQL script name. When you are asked if you want to save the changes, click **YES**.

Creating the SQL script SelProductById

To create the SQL script SelProductById:

1. Repeat the steps that are listed for the InsertProduct script in “Creating SQL script InsertProduct” on page 18.
2. Replace the script name, for example, with `SelProductById`.
3. Type the following SQL statement into your script that is open in the Editor view:

```
SELECT * FROM DSN8910.PRODUCT
WHERE PID = :PRODUCT_ID
```

Creating the SQL script UpdateProductById

To create the SQL script UpdateProductById:

1. Repeat the steps that are listed for the InsertProduct script in “Creating SQL script InsertProduct” on page 18.
2. Replace the script name, for example, with `UpdateProductById`.
3. Type the following SQL statement into your script that is open in the Editor view:

```
Update DSN8910.PRODUCT SET NAME = :productname
WHERE PID = :product_id
```

Creating the SQL script DeleteProductById

To create the SQL script DeleteProductById:

1. Repeat the steps that are listed for the InsertProduct script in “Creating SQL script InsertProduct” on page 18.

2. Replace the script name, for example, with `DeleteProductByID`.
3. Type the following SQL statement into your script that is open in the Editor view:

```
DELETE FROM PRODUCT
WHERE PID = :product_id
```

After you create all of your script files, your Data Development view is displayed, as shown in Figure 18.

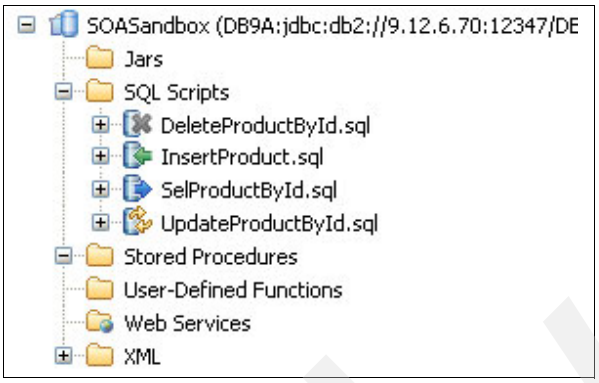


Figure 18 Created scripts in Data Studio Development Project view

Look at the symbols in front of the script names to see that they have a meaningful appearance. For example, the script that you will use to insert data into your Product table, using your Web service, has a green arrow pointing to a little disk volume symbol, and so on.

Creating a stored procedure

Your first Web service now consists of separate SQL scripts, but we also want to show you how to include existing stored procedures to access common table objects into your Web service.

To make sure that you do not run into any setup-related problems, we first guide you through creating a native SQL stored procedure. The main advantages of using an SQL stored procedure in our scenario are:

- You can easily create a native SQL stored procedure with just one SQL statement issued, for example, using SQL Processor Using File Input (SPUFI).
- A native SQL stored procedure runs within the DBM1 address space, and you do not have to set up a Workload Manager (WLM) environment and a related started task JCL.

Go to SPUFI, and execute the CREATE PROCEDURE statement, as shown in Figure 19 on page 22.

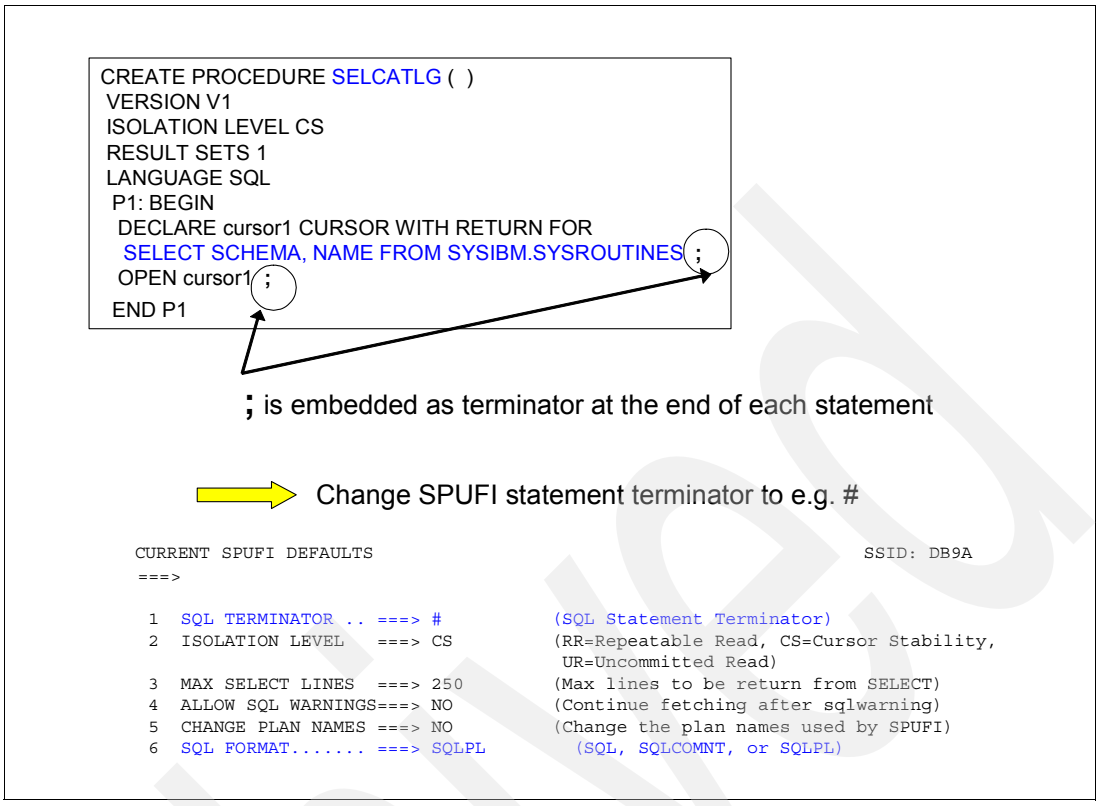


Figure 19 CREATE SQL stored procedure

The name of the procedure that you are going to create is SQLCATLG. The schema name is your current SQLID. After you have successfully executed the CREATE PROCEDURE statement, the procedure is ready for you to use.

You can test your stored procedure using Data Studio:

1. In the Data Source Explorer view of Data Studio, make sure that you are currently connected to your DB2 subsystem. You are successfully connected if the small icon that is next to your database connection is green. If this is not the case, right-click the database name, and click **Reconnect**.
2. Navigate down to the Schema level of the view, as shown in Figure 20 on page 23.

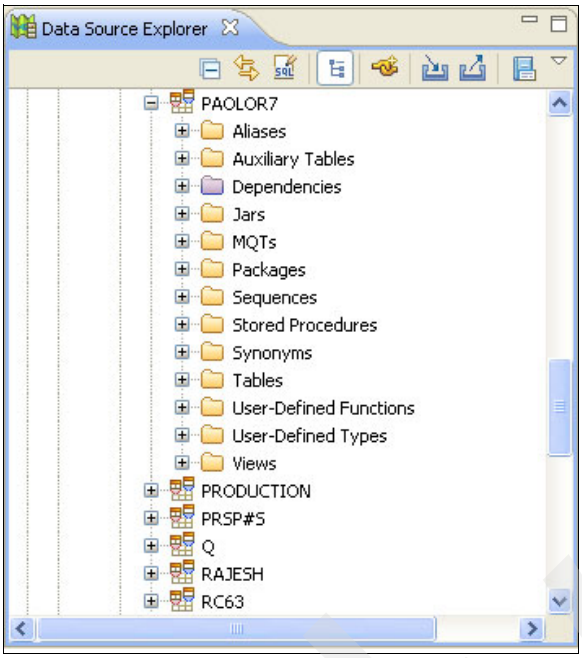


Figure 20 Data Source Explorer: Schema explored

3. Within the Schema, identify the schema name that you used to create your stored procedure, and navigate through the directory to further explore the information.

   We used the schema name PAOLOR7. Figure 21 on page 24 shows all types of objects that are stored under this schema name. We are looking for Stored Procedures. Navigate through the directory to further explore the available stored procedures.

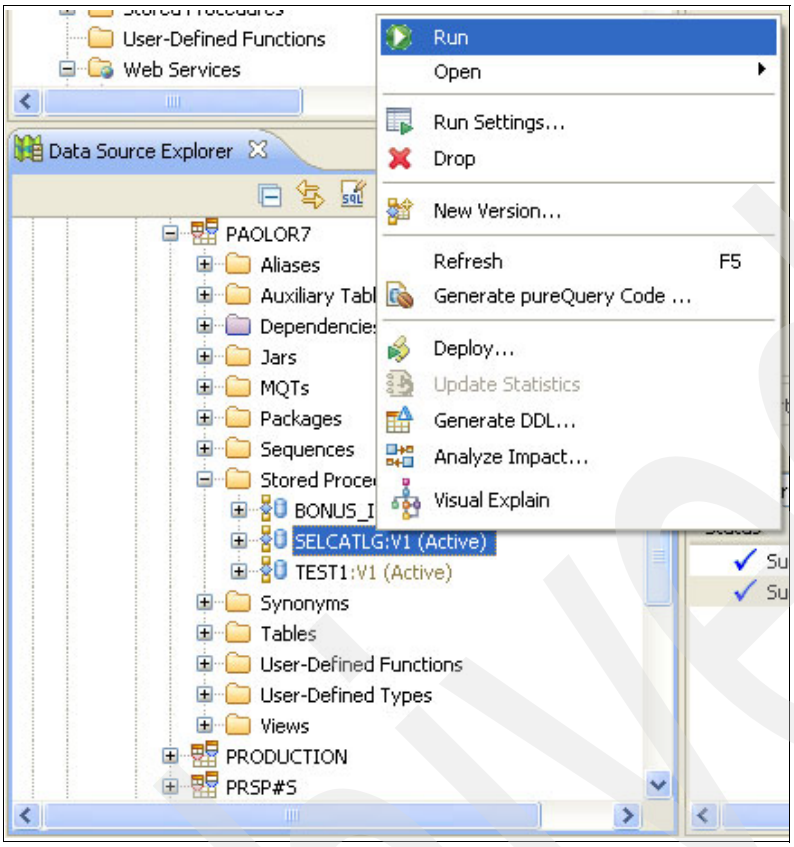


Figure 21 Data Source Explorer: Schema view

4. Right-click the procedure that you just created, and select **Run**.
5. In the lower-right corner of your Data Studio application, the result of your stored procedure is displayed in the Results Output view, as shown in Figure 22 on page 25.

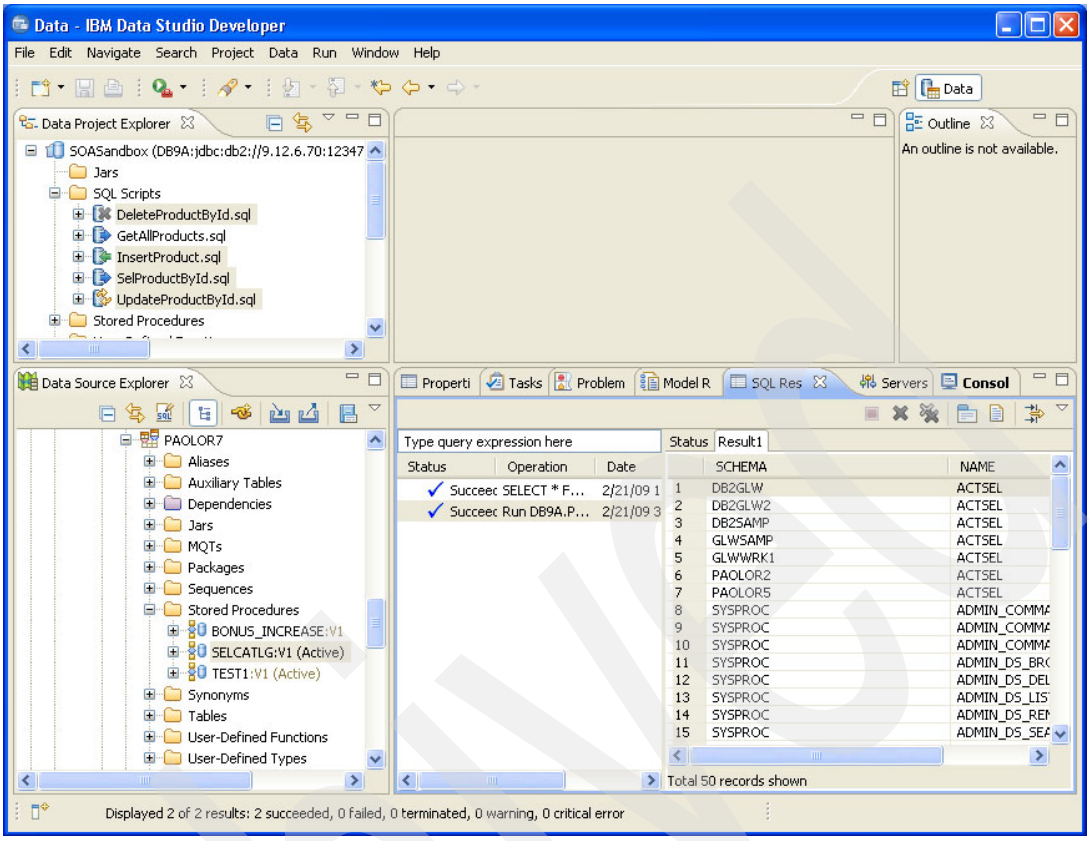


Figure 22 RUN stored procedure results in Result view

Creating your first Web service: Simple case

You have now created four SQL scripts and one stored procedure. Together, they are supposed to form one Web service.

Use these steps to create your first Web service. We provide detailed instructions in the following sections:

- "1- Defining a new Web service in your Data Project Explorer" on page 25
- "2- Populating the new Web service (drag-and-drop)" on page 26
- "3- Validating your SQL scripts and your stored procedure" on page 28
- "4- Building and deploying a Web service" on page 29

1- Defining a new Web service in your Data Project Explorer

You created new SQL scripts using your Data Project Explorer. Next, you start using the Web Services part:

1. Click **Web Services**, and select **New**. A New Web service window is displayed, as shown in Figure 23 on page 26.

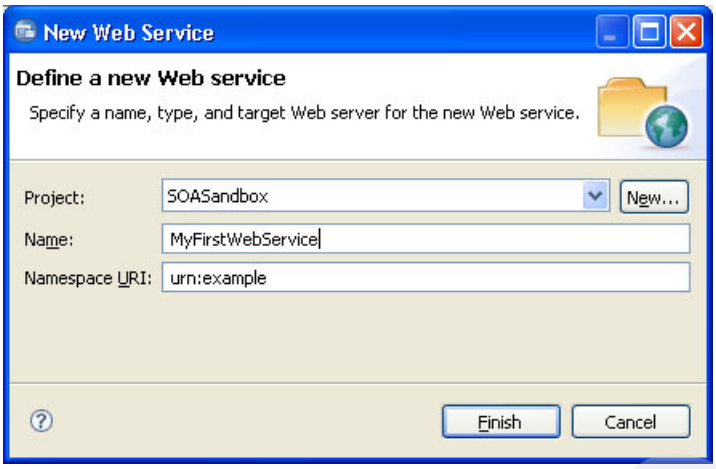


Figure 23 New Web service definition in Data Project Explorer

2. In the Project field, accept the default Project. It is `SOASandbox` in our example.
3. In the Name field, type a name for your first Web service. We called it `MyFirstWebService`.
4. In the Namespace URI field, accept the default. More information about the namespace use and why it is important in real-life applications can be found in *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.

2- Populating the new Web service (drag-and-drop)

You only created an entry within your Data Project Explorer Web Service section. Nothing else has happened so far. Now, you must decide which SQL scripts and stored procedures you want your Web service to consist of. In our example, we mentioned that the first Web service consists of the four SQL scripts and the SQL procedure.

To connect the SQL scripts and the SQL procedure to your Web service:

1. In the Data Project Explorer, open the SQL scripts, select the four recently created SQL scripts, and drag-and-drop all four of them to your newly created Web service name, as shown in Figure 24.

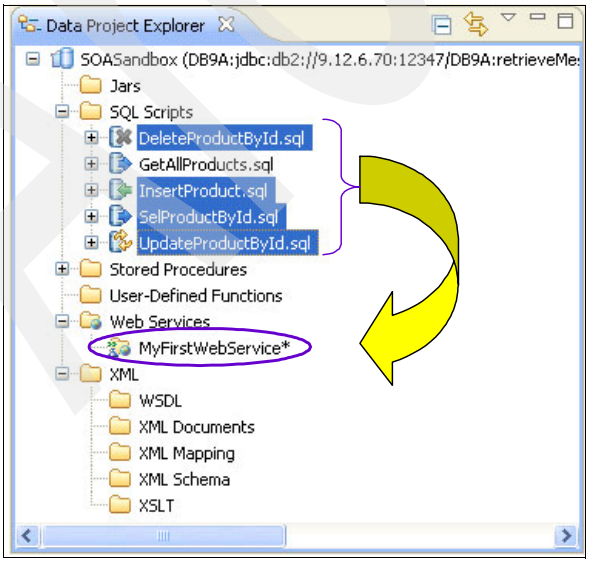


Figure 24 Drag-and-drop SQL scripts to Web Service

2. In the Data Source Explorer, open your schema to the list of available stored procedures, select your recently created stored procedure, and drag-and-drop this procedure to your newly created Web service name, as shown in Figure 25.

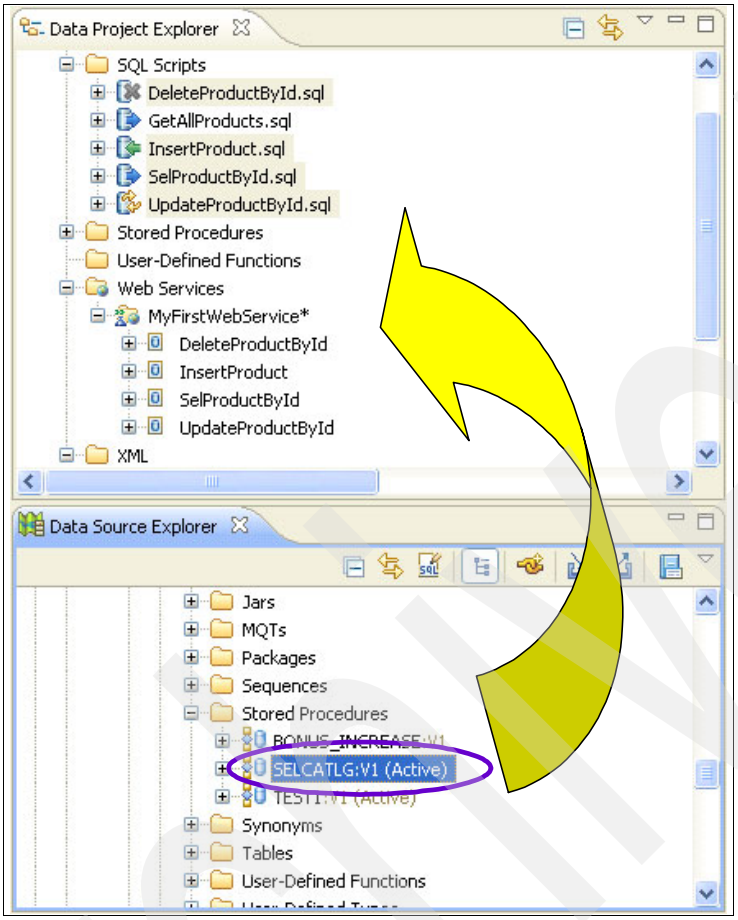


Figure 25 Drag-and-drop your stored procedure to the Web service

In your Data Explorer view, review the results. You should now see the four SQL scripts and the SQL procedure under your Web service name, as shown in Figure 26 on page 28.

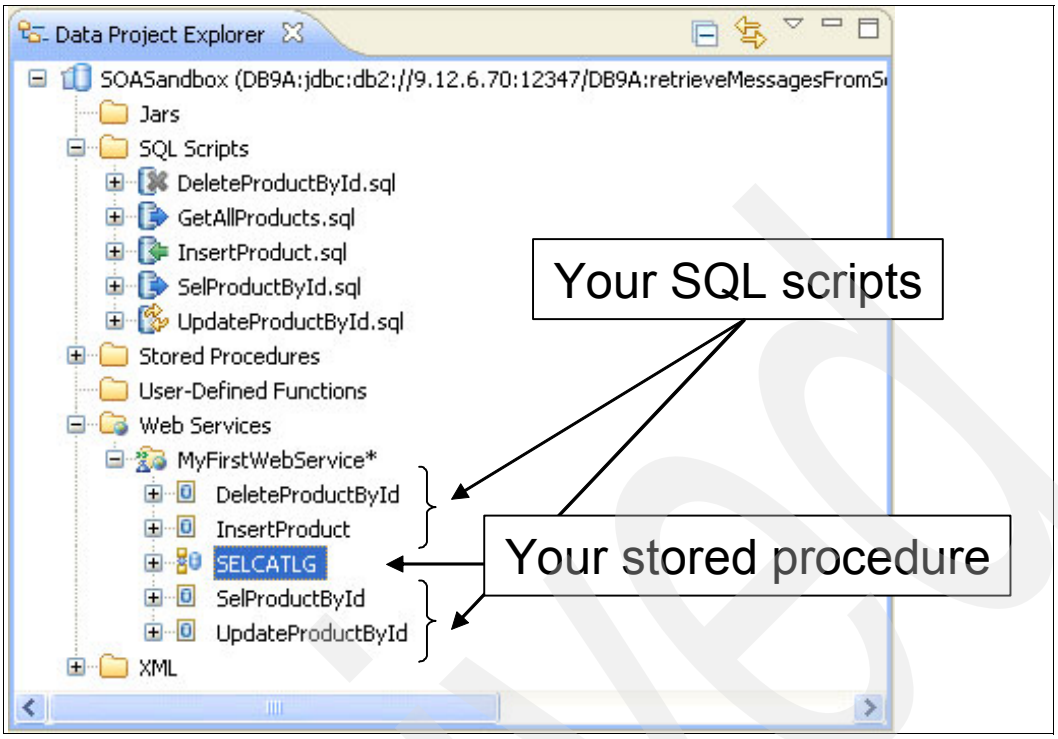


Figure 26 Populated Web service

3- Validating your SQL scripts and your stored procedure

Before you start to build your Web service, you might want to validate the syntactical and contextual correctness of your four SQL scripts and your stored procedure.

To validate your SQL scripts and stored procedure:

1. Click each component of your future Web service, one at a time, and click **Edit**.
2. In the Editor mode, click **Validate**, as shown on Figure 27.

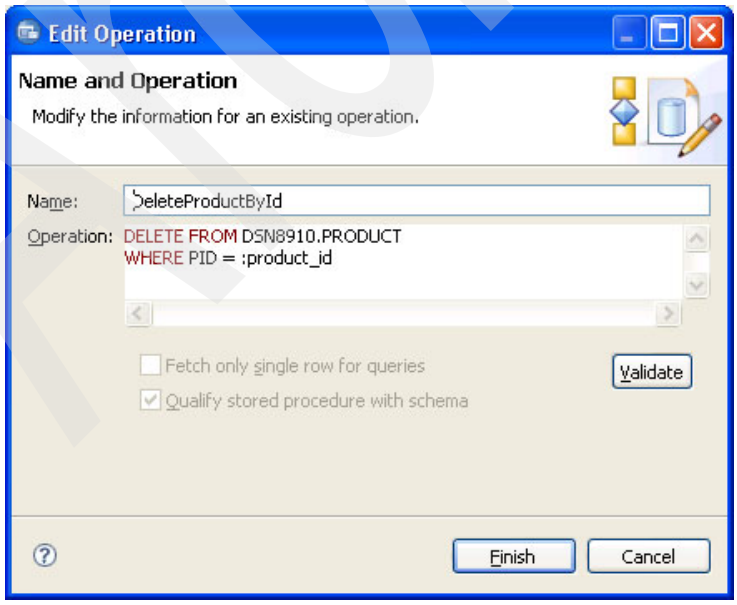


Figure 27 Validate Web service components

The **Validate** button invokes the SQL parser validation routine, which attempts to determine if the text entered in the Operation text area is a valid SQL statement or not.

3. Click **Finish**. The contextual validation occurs. In our example, Data Studio recognizes an SQL error in the SQL script DeleteProductById, as shown in Figure 28. The problem here is that database object DSN891.PRODUCT does not exist. The correct database object name is DSN8910.PRODUCT. We correct the problem and successfully repeat the steps for all other components.

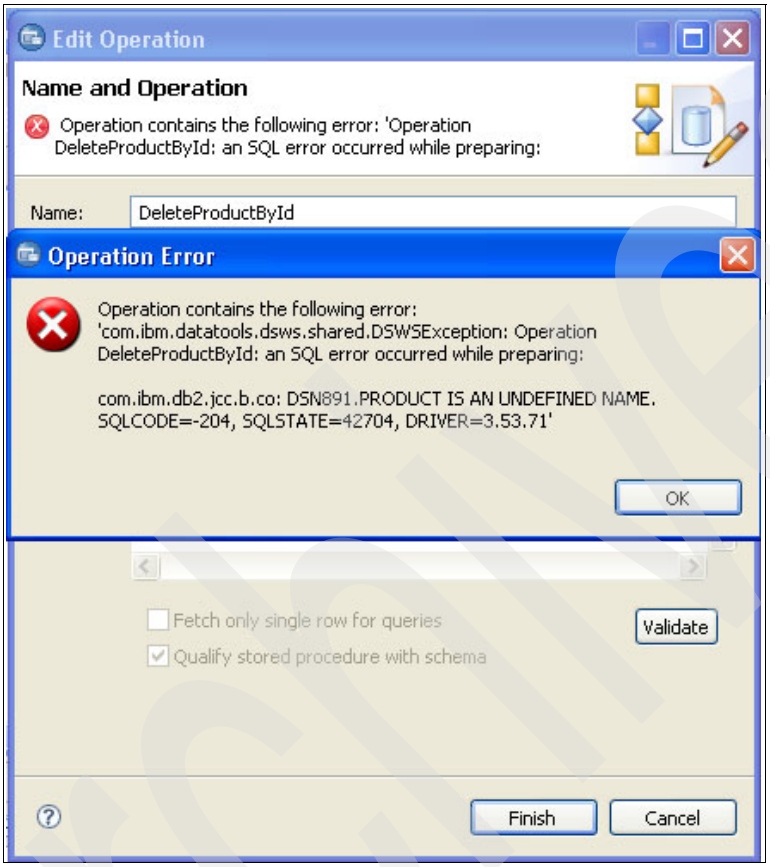


Figure 28 SQL error in Web service component SQL script DeleteProductById

4- Building and deploying a Web service

We gave the Web service a name and associated the SQL scripts and SQL procedure to it. These two steps are not enough to make the Web service a Web service. In this section, we show you how you can actually create a Web service and deploy it with Data Studio. We show the iterations of the full process.

Creating a new Web service: Simple case

To create a new Web service:

1. Go to your Database Project explorer, and navigate through your Web service until you see your Web service name under the Web services section.
2. Right-click the Web services name, which is `MyFirstWebService` in our example, as shown on Figure 29 on page 30.

> **Tip:** Look at Figure 29. The (*) asterisk behind the Web service name indicates that the Web service has not yet been built.

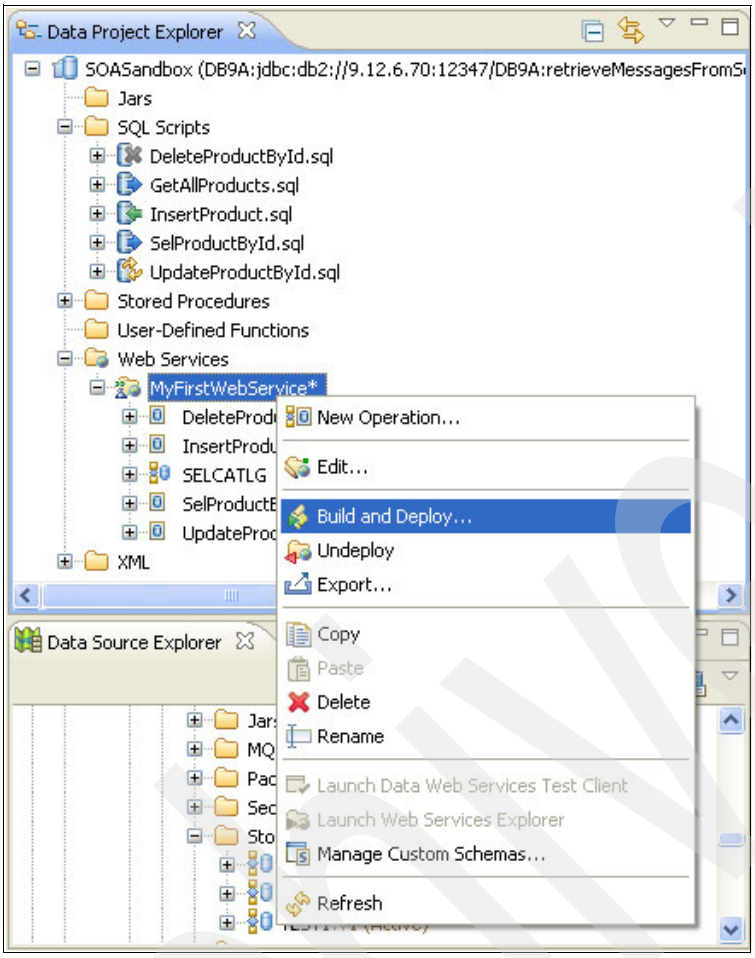


Figure 29 Build and Deploy Web service menu

3. Click **Build and Deploy**. A new window is displayed, as shown in Figure 30 on page 31.

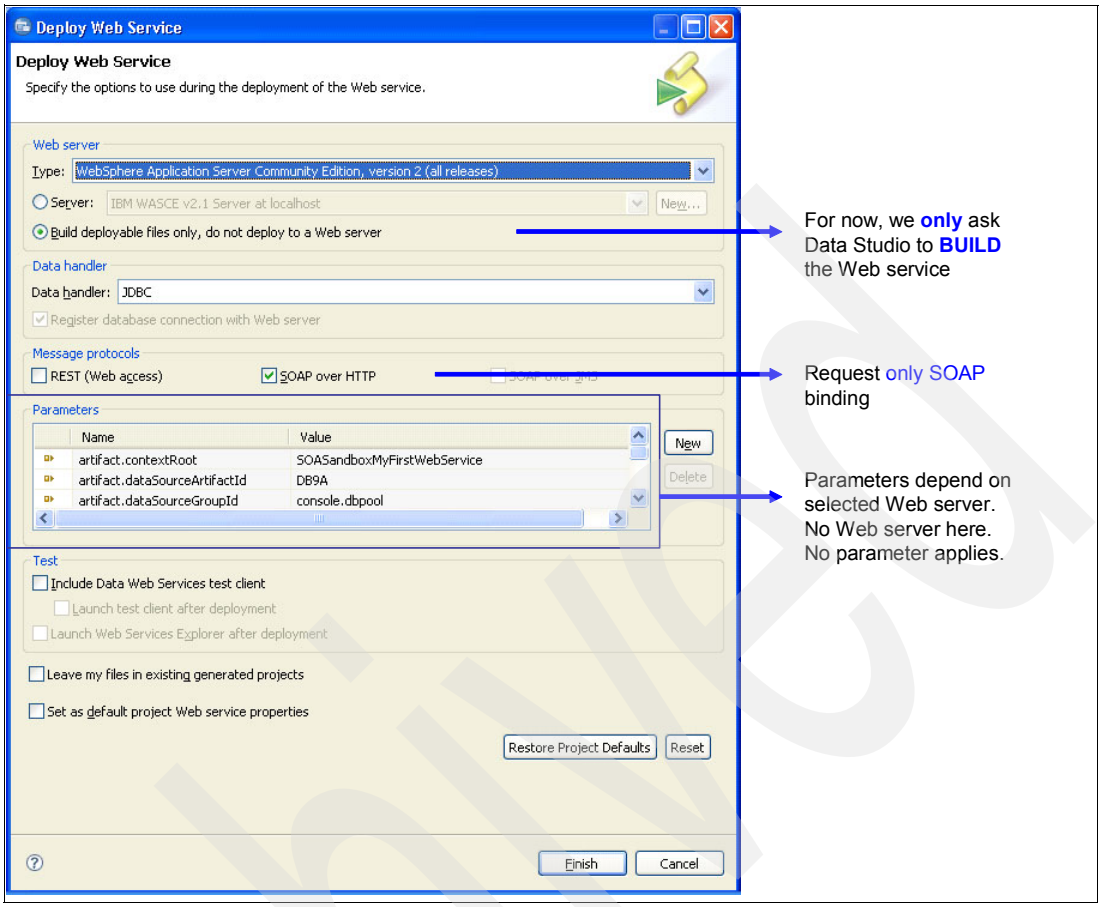


Figure 30 Deploy Web Service window

In the Deploy Web Service window, in our example, we make the following selections:

- Web server: **Build deployable files only, do not deploy to a Web server**

  So far, you have only installed WebSphere Application Server Community Edition on your workstation. You have not yet installed the WebSphere Application Server Community Edition in Data Studio. As a consequence, even if you selected the Server option here, no Server displays in the pull-down menu for Server; therefore, you cannot use this option.

- Data handler: **JDBC**

  The given options here are JDBC and pureQuery (static, no bind), and pureQuery (static, with bind). pureQuery[1] is a new large topic that we do not discuss in this book, because we choose JDBC.

- Message protocols: **SOAP over HTTP**

For our first Web service, we create SOAP message-oriented Web service files:

- Parameters: **default**

[1] pureQuery gives database application developers an easy, GUI-based means to significantly increase productivity in both the design and implementation phases through the user-initiated automatic transformation of relational data into Java objects for access and manipulation of data. pureQuery's functionality eliminates traditional Java Database Connectivity (JDBC) programming by integrating the query language with Java itself. Refer to http://www.ibm.com/developerworks/db2/library/techarticle/dm-0708ahadian/ and related articles.

The parameters here depend on which Web server you use for storing your Web services. We have not selected a server yet, so we do not worry about these settings.

– Test: no option selected

  We discussed testing the Web service, and we will discuss the options for tooling support later in this Redpaper.

– Set as default project Web service properties: do not select this option.

  We need to explain different settings in this Redpaper; therefore, we do not want the current selection to be the default for later Web service builds.

4. Click **Finish** to build your first Web service. While the system builds your Web service, multiple windows display and disappear. When the system completes all of the necessary steps, no more windows are displayed, and, assuming that your Web service built successfully, the * (asterisk) next to your Web service name, as shown in Figure 26 on page 28, disappears.

Congratulations. You have just created your first Web service using SOAP over HTTP message binding.

You are probably now asking the following questions: Where do I see this Web service now? What can I do with it? How can I invoke this service and from where do I invoke it? How does the result look, and what can I do with this result?

To learn the answers to these questions, continue reading.

Checking the results of your first Web service

After you have successfully built your Web service, the information that you can see in your Database Project view is not much different from what you saw before. The only difference is that the * (asterisk) next to the Web service name has disappeared.

To continue working with what was built for you, you must now add another view to your Data Studio window.

On your Data Studio Menu, go to **Window** → **Show View** → **Navigator**, and press Enter.

A Navigator view appears on your Data Studio window; however, it might or might not appear exactly at the same location as it appears in Figure 31 on page 33. If your Navigator shows up at another location, and you want your window to look like what we show you, drag the top of the windows and drop it in the upper left corner, where you also see your Data Project Explorer.

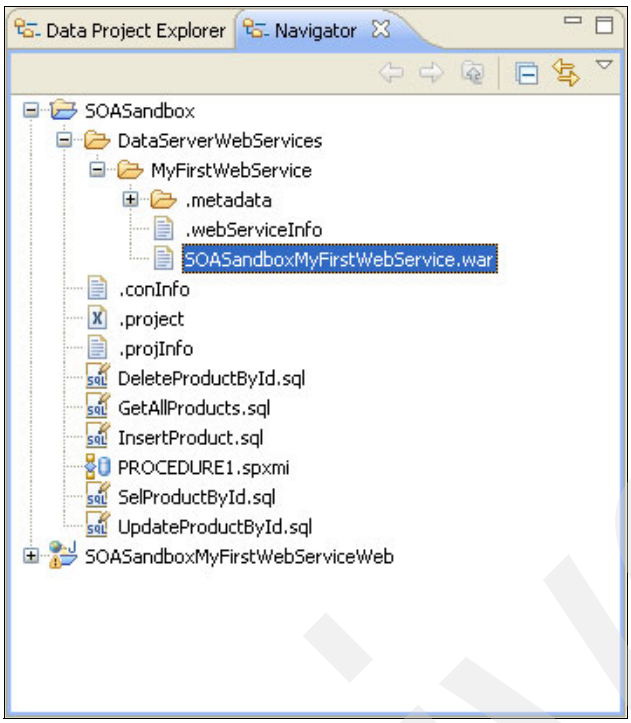


Figure 31 Navigator view in Data Studio

Notice in Figure 31 that several files are listed under your Web services folder, MyFirstWebService.

All the files that are actually making up the Web service are contained in the SOASandboxMyFirstWebService.war file. *Web archive* is what *.war* represents. The .war file is a compressed file that consists of several other files. If you try to open the .war file in the Editor view, the contents are unreadable. To extract the .war file, export it to your workstation, and use any appropriate program to decompress it.

To export the .war file to your workstation:

1. Right-click the .war file, and select **Export**. A new window is displayed.
2. Expand the **General folder**, select **File System**, and click **Next**. A new window is displayed.
3. In the middle of the window, you can click **Browse** to select the correct folder for the To Directory.
4. Select the directory to which you want to export your .war file, and click **Finish**.
5. Use Windows Explorer to identify the .war file, and double-click it. If you have a program that decompresses files installed on your workstation, you can now look at the contents of your .war file. You see seven separate files as shown in Figure 32 on page 34.

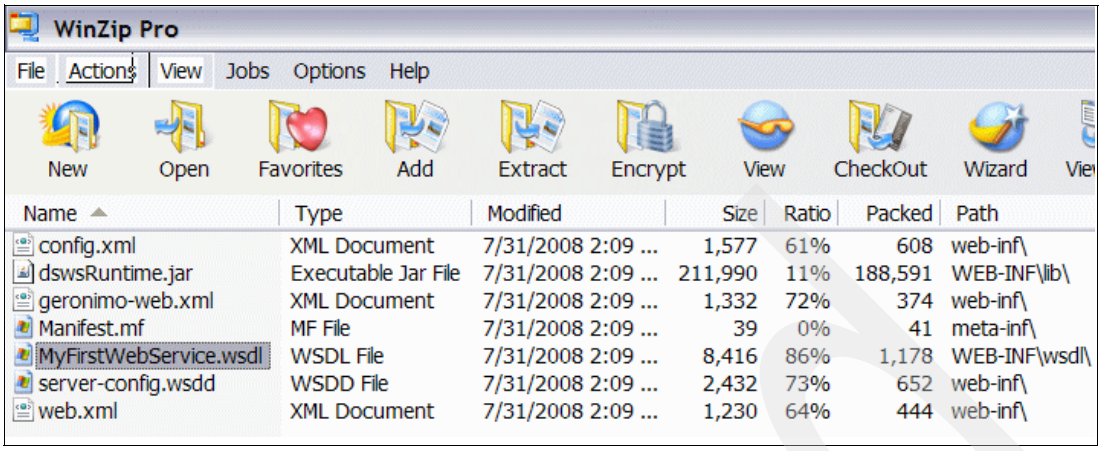


Figure 32 SOASandboxMyFirstWebService.war file contents

The generated files are the *service artifacts*, which is the term that is used often in Data Studio.

For a detailed description of the contents of the .war file, refer to *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.

Concluding the simple case

You have just built a new Web service. The build step completed successfully. If you want to run your Web service, you must manually install the .war file on a J2EE-compliant Web server, which is not a convenient and suitable sandbox environment for a DB2 systems programmer. Therefore in our next iteration, we instead use the WebSphere Application Server Community Edition.

Publishing the new Web service

Now that you created your first simple Web service, we look at a more advanced case where we show how to publish our Web service in a WebSphere Application Server Community Edition J2EE server. The steps that we describe are:

1. Adding a WebSphere Application Server Community Edition instance within Data Studio
2. Building and deploying your Web service to the Web server

Adding a WebSphere Application Server Community Edition instance within Data Studio

If you have followed our instructions, WebSphere Application Server Community Edition (WSCE) is already installed on your workstation. If you omitted this step, return to “Getting WebSphere Application Server Community Edition” on page 9, and perform the provided steps. Having a ready-to-run WebSphere Application Server Community Edition installed on the workstation is necessary to provide a lightweight J2EE server for testing Web services.

Note: In case you skipped the WebSphere Application Server Community Edition section, you can download a no-charge version of WebSphere Application Server Community Edition from:

`http://www.ibm.com/developerworks/downloads/ws/wasce/`

To add the WebSphere Application Server Community Edition instance to your Data Studio:

1. From the Data Studio menus, select **Window** → **Show View** → **Other** → **Server**, and click **Enter**. A new window is displayed.
2. Expand **Server** (if not already done), and click **Servers**.

   A new tab named Servers appears in the lower right corner of your Data Studio window, as shown in Figure 33.

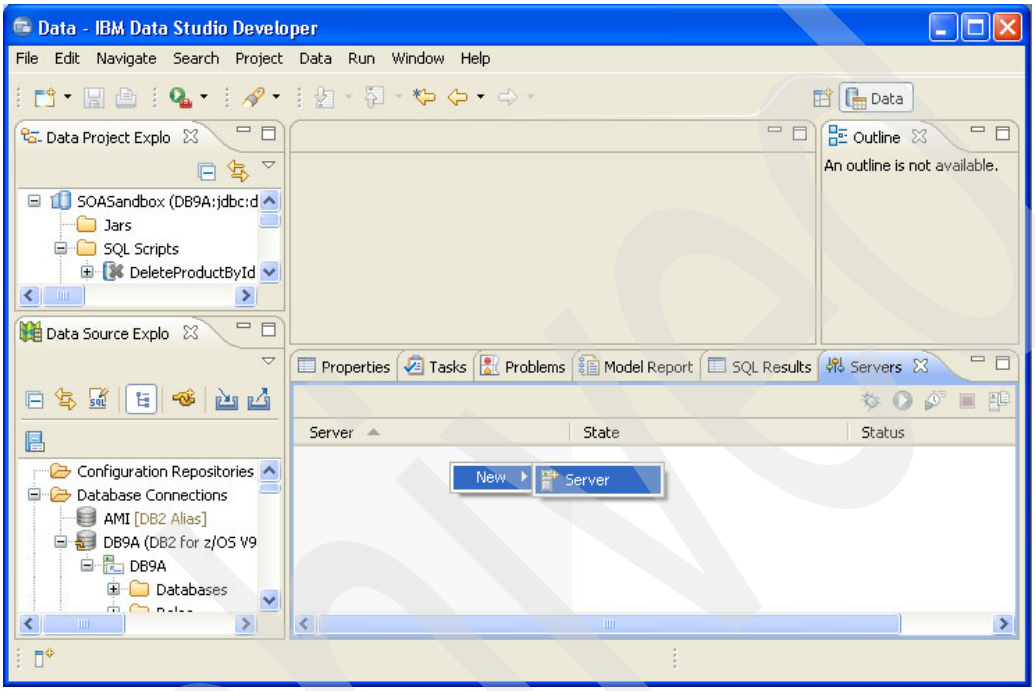


Figure 33 Add new Server instance to Data Studio

3. Click in the whitespace, and you see the little window that is already shown in Figure 33. Select **New** → **Server**. Another window is displayed, as shown in Figure 34 on page 36.
4. You can accept all preset selections, as shown on Figure 34 on page 36. The Server's host name is `localhost`, because it resides on your machine. Select **Next**.

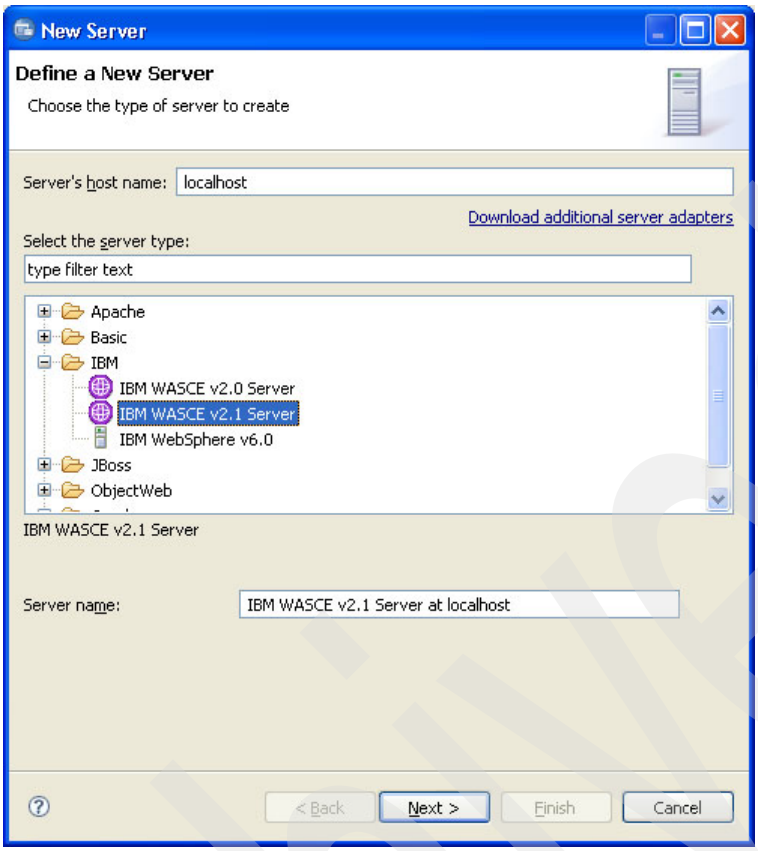


Figure 34 Add WebSphere Application Server Community Edition instance to Data Studio

5. If you have not yet configured a WASCE Runtime, you need to configure it in the next window, as shown in Figure 35 on page 37. You are asked to provide a Java Runtime Environment (JRE™) and the absolute path to the WebSphere Application Server Community Edition installation. We select the default workbench JRE, which comes with Data Studio Developer V2.1. You get a warning, because this version is a 1.6 JVM™ and WASCE V2.1 is only certified for the 1.5 JVM, but you can ignore the warning.

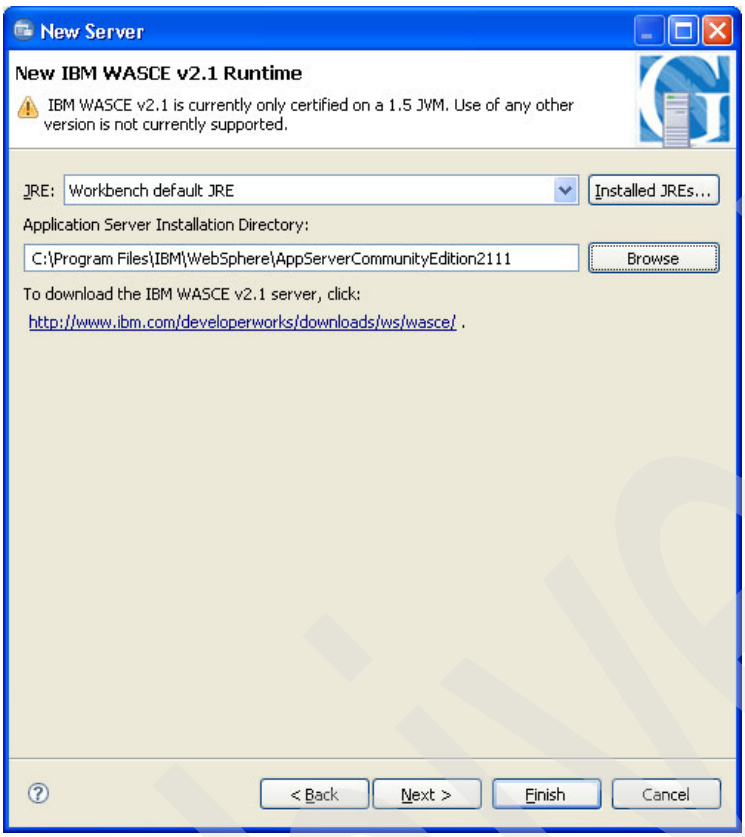


Figure 35 Configuration of a new WASCE Runtime

6. The next window is already populated for you as shown in Figure 36.

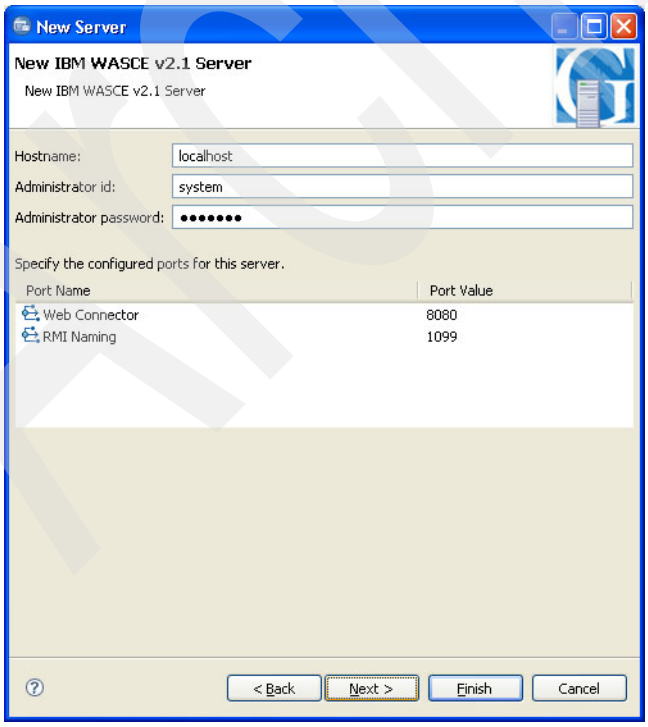


Figure 36 WebSphere Application Server Community Edition port definition in Data Studio

The Administrator ID and Administrator Password are the credentials of the WASCE admin user. By default, the Administrator ID is *system* and the password is *manager*. You might need the Administrator ID and Administrator Password at a later time when you try to log on to the Administration console from within or outside of Data Studio. The Web Connector defines the TCI/IP port for the HTTP protocol, which is 8080, by default. The Remote Method Invocation (RMI) Naming defines the port that is used by Data Studio to perform administrative tasks at the application server. This port is 1099, by default. If the port is occupied by another process, you might need to change it as described in “Resetting WebSphere Application Server Community Edition ports from Data Studio” on page 72.

7. Click **Finish**. You have successfully added the WebSphere Application Server Community Edition instance to your Data Studio, and the server definition also appears in the lower right corner of your Data Studio window, as shown in Figure 37.

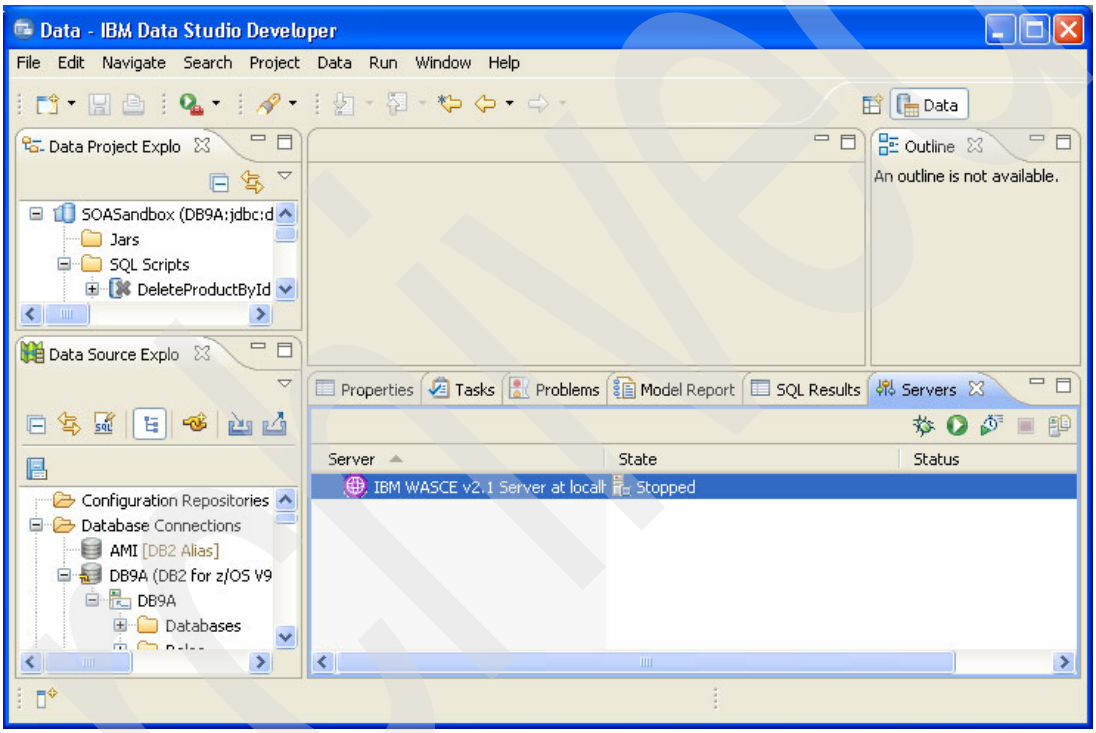


Figure 37 Defined WebSphere Application Server Community Edition instance in Data Studio

You took the prerequisite steps so that Data Studio cannot only build your Web service, which we successfully performed in the first approach, but Data Studio can now also deploy the .war file to your WebSphere Application Server Community Edition server. We show this task in the next step.

Building and deploying your Web service to the Web server

We now repeat a couple of steps that you have already performed when you created your first Web service. The goal is to build your Web service again, but also deploy it to your Web server so that upon completion, you can use the tooling that is built into Data Studio to also test the Web service for the first time.

To build and deploy your Web service:

1. Go to your Development Project view. Make sure that you are no longer in the Navigator view. Right-click your Web service, and select **Build and Deploy**. If you need visual assistance for this step, you might want to go back to Figure 29 on page 30.

The Deploy Web Service window is displayed.

2. On the Deploy Web Service window, make a few other choices, as shown on Figure 38.

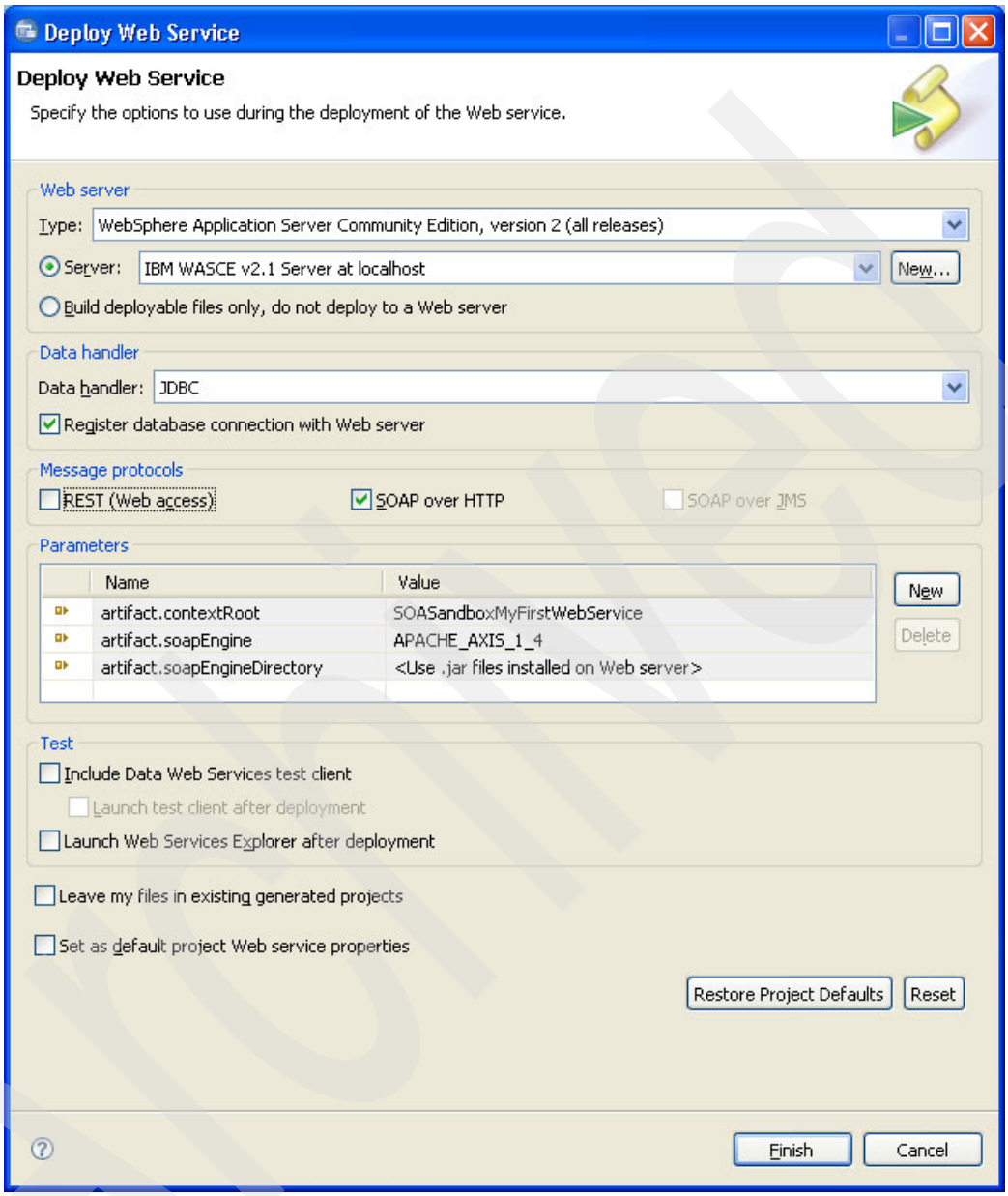


Figure 38 Deploy Web Services window where you deploy to server

The choices are:

a. Select **WebSphere Application Server Community Edition version 2 (all releases)** as the Web server type.

b. Select **Server**, and choose **IBM WASCE v2.1 Server at localhost**.

c. Keep only the default option **SOAP over HTTP** binding for now. We show you other bindings later in this Redpaper.

As mentioned, do not focus on the Parameters now, because those parameters relate to a specific server type, and we have not selected a server yet.

For our sandbox examples, the parameters are not important for us. We have added them with their explanations in Table 1 for completeness. If you follow our specific example, you can omit reading Table 1.

Table 1 WebSphere Application Server Community Edition parameters

Parameter	Description
artifact.contextRoot	Specifies the top-level directory of the application for a Web service when it is deployed to a Web server.
artifact.dataSourceGroupId	Specifies the directory or directory tree in the WebSphere Application Server Community Edition repository where the unique artifact directory is located.
artifact.dataSourceArtifactId	Specifies the name of the database connection pool. This parameter is case-sensitive. This value is the prefix of the library's file name.
artifact.dataSourceVersion	Specifies the version number that you choose to be appended to the file name together with a .jar extension to create the library file name.
artifact.soapEngine	Specifies the SOAP engine to us, if you selected a SOAP message protocol.
artifact.soapEngineDirectory	Optional parameter that specifies the directory on the local file system with the JAR files for the SOAP engine. If you specify this directory, the workbench packages the JAR files in the WAR file that you generate when deploying the Web service.

d. Do not select anything in the Test boxes.

3. Click **Finish**. Several windows display, but this time it might seem as though the process is hanging somehow, because the WebSphere Application Server Community Edition server is now being started, if it was not started before. You requested it in Figure 38 on page 39. You specified that you want Data Studio to not only build the Web service but also to deploy it onto the server. Figure 39 on page 41 shows the startup process for WebSphere Application Server Community Edition in the lower right corner of the Data Studio window.

   After Data Studio finishes its work, the Web service is deployed to your WebSphere Application Server Community Edition server.

   You might recognize that in the lower right corner the WebSphere Application Server Community Edition server now shows as started. In Figure 39 on page 41, we saw that your deployment request initiated the start-up of the server. Now it stays up until you either stop it or you close the Data Studio application. As a consequence, all subsequent build and deploy requests can use the server that is currently running.

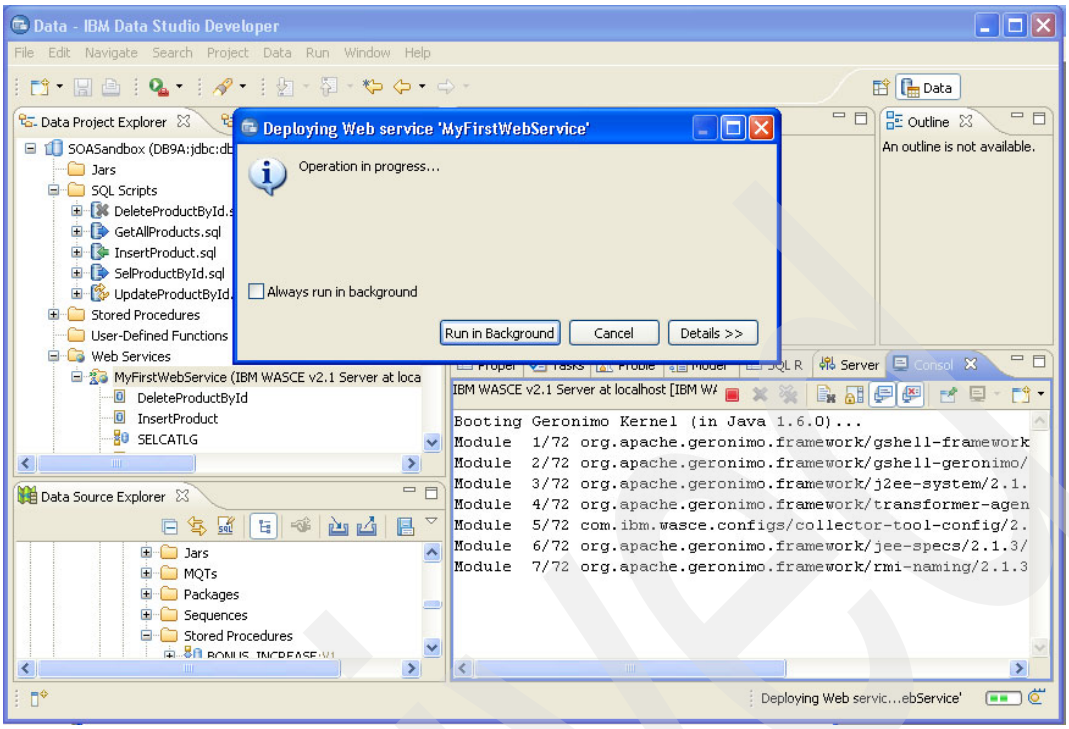


Figure 39 Starting WebSphere Application Server Community Edition server during build and deploy

Now that you have added the WebSphere Application Server Community Edition instance, which is your J2EE server, to Data Studio, and you used the additional option to deploy the newly created Web service to this server, you are ready to test your new Web service for the first time. Figure 40 summarizes the process.

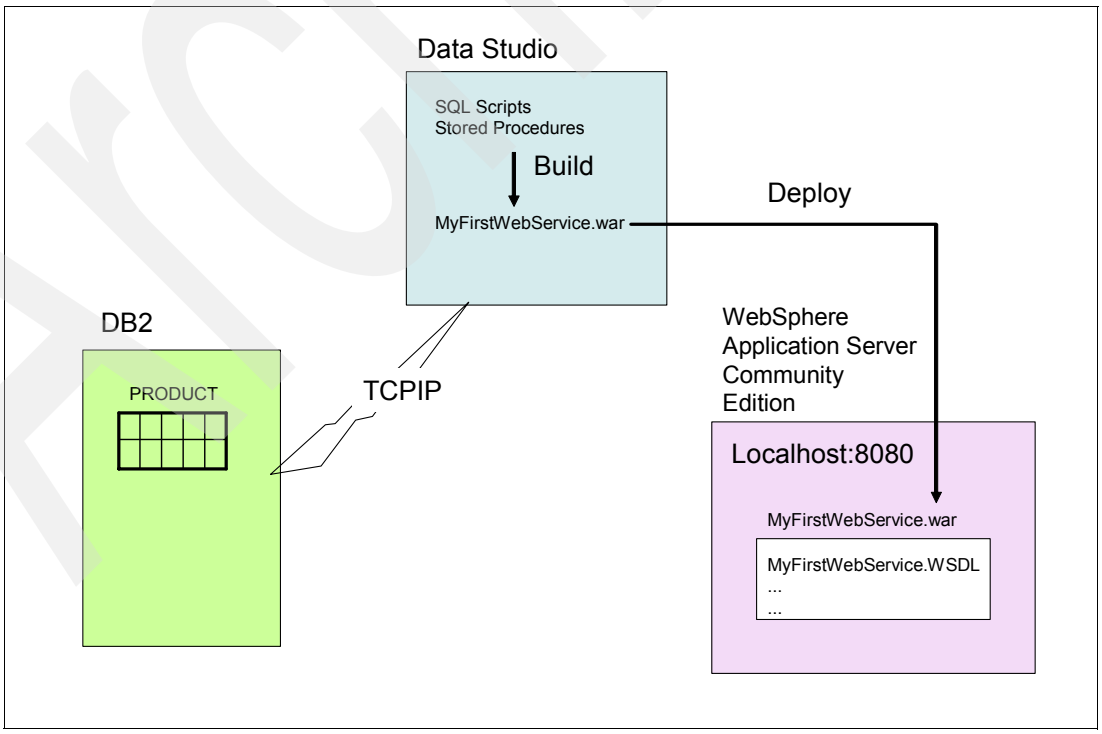


Figure 40 Overview picture of build and deploy

Testing your Web service

There are many ways to test your new Web service; one easy way is to use the built-in Web Services Explorer of Data Studio, which is what we are going to show in this section.

Using the built-in Web Services Explorer

To use the built-in Web Services Explorer:

1. Go to the **Data Project Explorer** view, open your project, and explore your Web service.
2. Click your Web service name and click **Launch Web Services Explorer** to start the Web Services Explorer in Data Studio, as shown in Figure 41.

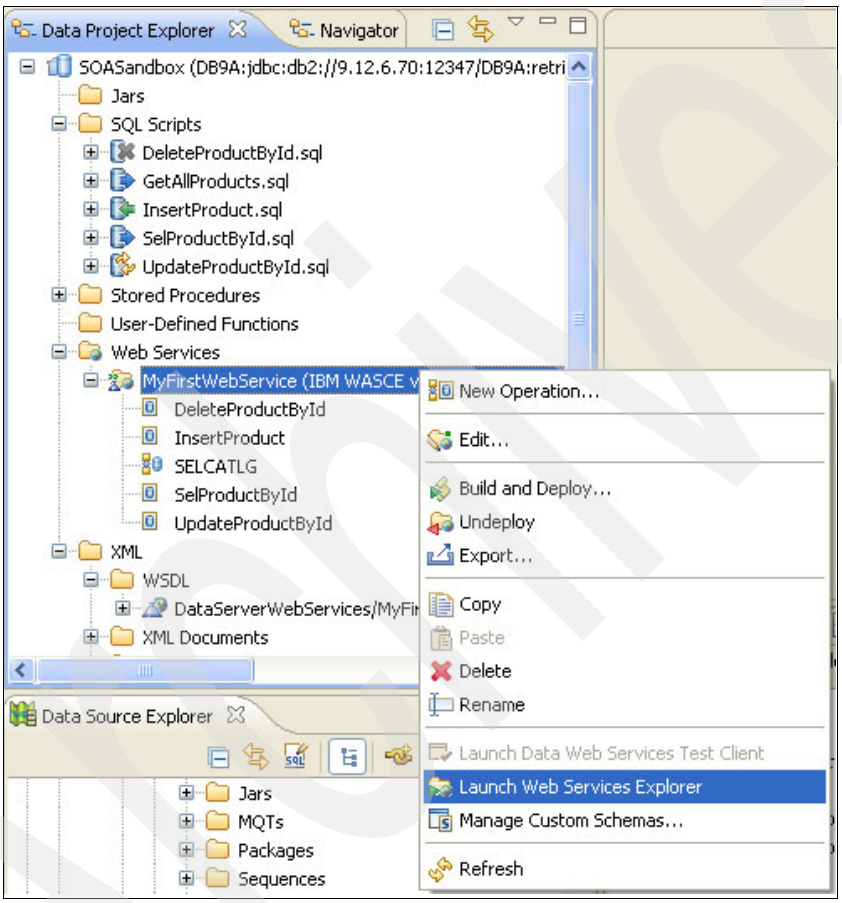


Figure 41 Manually launch Web Services Explorer

Figure 42 on page 43 shows the Web Services Explorer on Data Studio window.

Note: For better readability, we show just the upper right part of Figure 42 (the Web Services Explorer within Data Studio) on the next few visuals.

The Web Services Explorer is displayed in the upper right view of your Data Studio window, as shown on Figure 42 on page 43.

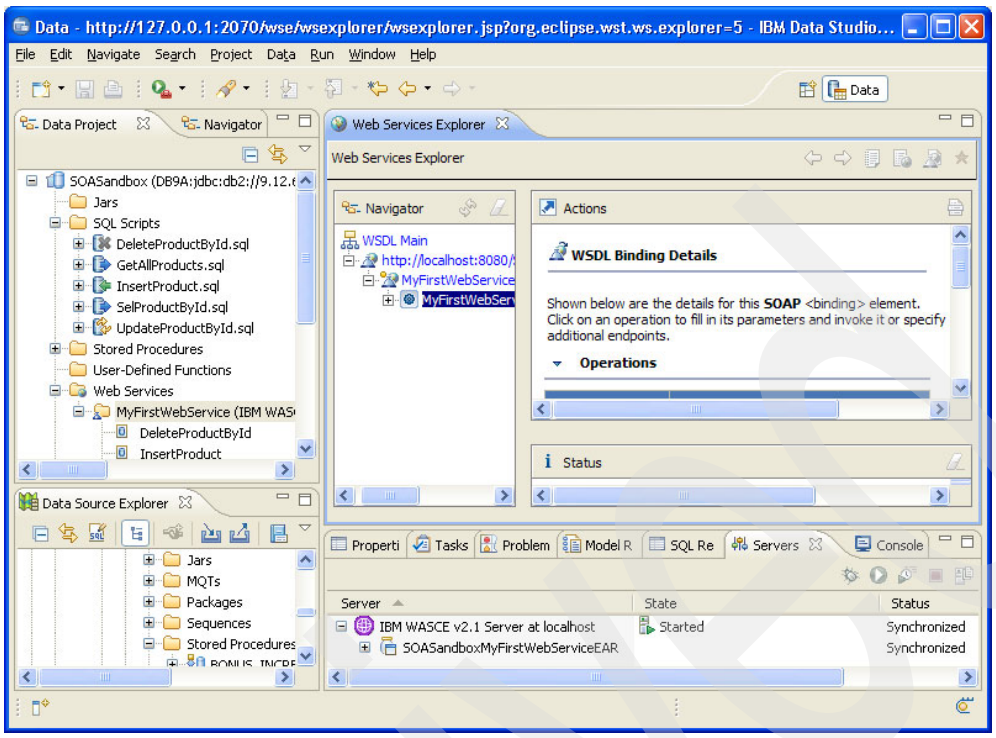


Figure 42 Web Services Explorer on Data Studio window

Note: If the WebSphere Application Server Community Edition has stopped by the time that you attempt to click **Launch the Web Services Explorer**, this option is not highlighted and is not selectable in the context menu that is shown in Figure 41. In this case, refer to the lower right corner of your window where, usually, the server instance is shown (refer to Figure 42). In Figure 43, the server is currently stopped. To start it, click the circled arrow in the command list.

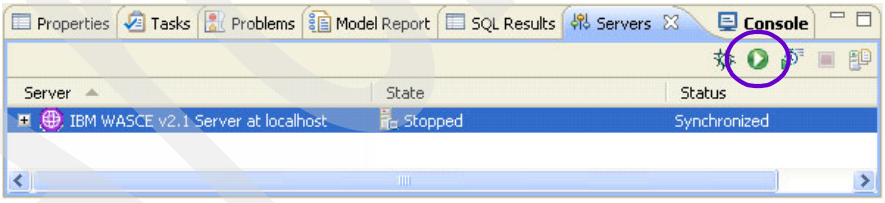


Figure 43 Manually start WebSphere Application Server Community Edition from Data Studio

Figure 44 on page 44 shows a more detailed view of the Web Services Explorer. On the left side, there is a detailed list of all of the components that form your Web service. In our case, there are four SQL scripts and the stored procedure SQLCATLG.

The endpoint to which the arrow points in Figure 44 on page 44 is the location where the deployment process put the WSDL for this Web service.

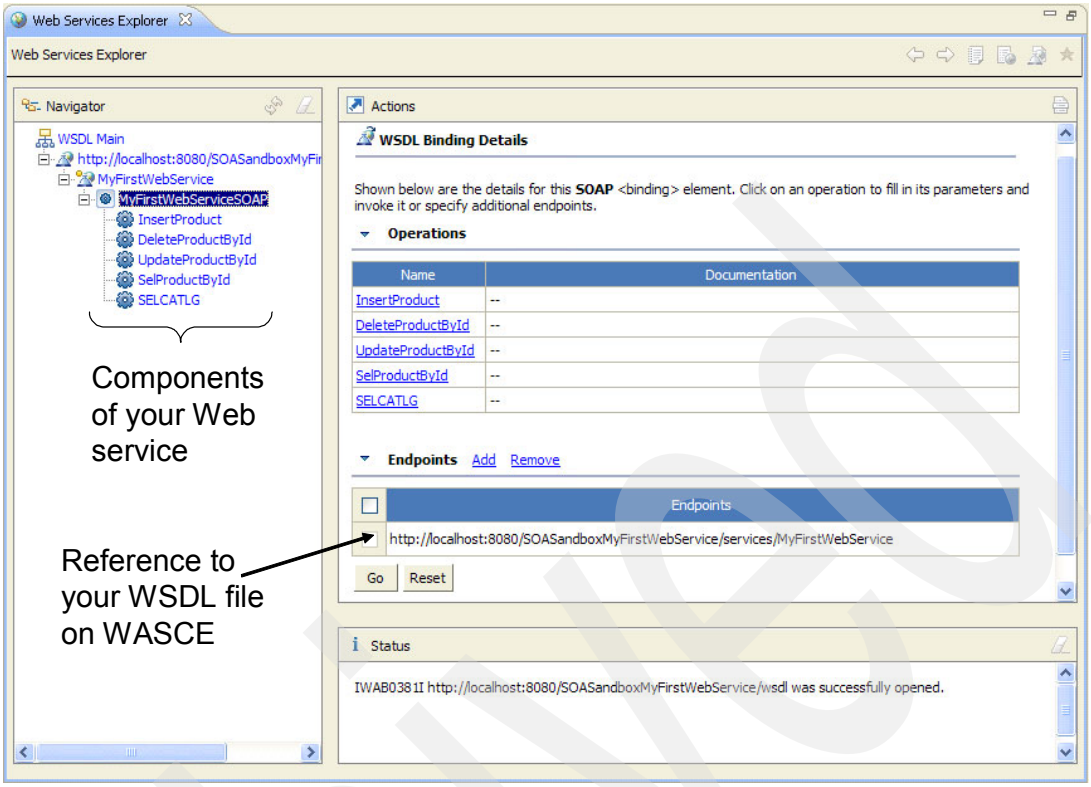


Figure 44 Web Services Explorer

Testing the InsertProduct component

Now, it is time to actually test the components of your Web service and see how the results are presented. Let us assume that the Product table to which you refer in your SQL statements is currently empty. As a consequence, it is good to start with the InsertProduct component:

1. On the left side of the Web Services Explorer, click **InsertProduct**. The appearance changes to what is shown in Figure 45 on page 45.
2. Populate the values on the right side of Figure 45 on page 45. We used the following values for our example:
   - product_id: `123`
   - productname: `Desk Chair, white`
   - price: `49.99`
   - Select **nil** for all other values.
3. Scroll down, and click **GO** to execute this component of your Web service.

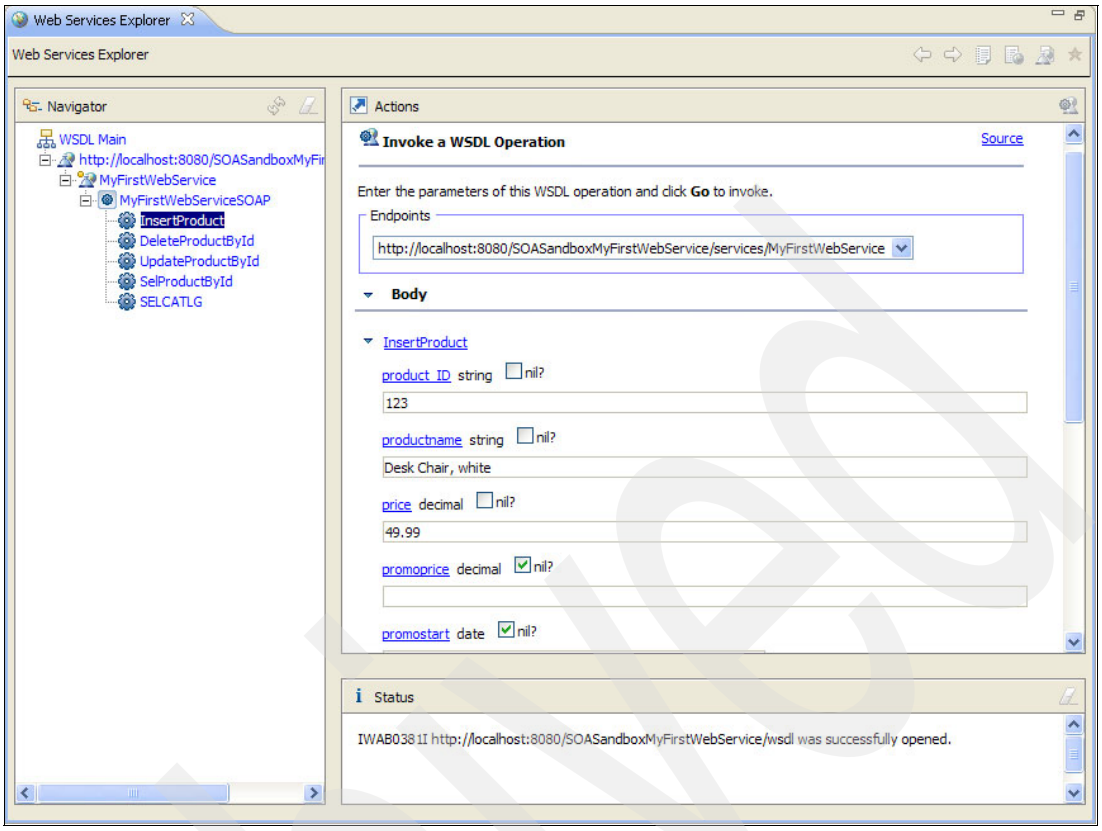


Figure 45 InsertProduct test in Web Services Explorer

4. To check the result of your Web service execution, you can refer to the Status section on the lower right area of your window. You get the response that is shown in Figure 46.

Figure 46 Result Form after the invocation of InsertProduct component

All you can tell from looking at this result, which is called the Result Form, is the fact that you inserted one row into your table using the InsertProduct component of your Web service, which was deployed to your WebSphere Application Server Community Edition and referred to through the endpoint name that we have shown in Figure 44 on page 44.

In the upper right corner of the Status part that is shown on Figure 46, if you now click **Source**, you get much better insight into how the Web service actually was invoked and executed. All Web service messaging is encoded in XML. In our example, as shown in Figure 38 on page 39, we selected to use SOAP binding; therefore, the format of the Web messages that are used whenever we invoke our Web service is SOAP.

Figure 47 on page 46 shows the SOAP messages that were used to execute the Web service when you clicked **GO**.

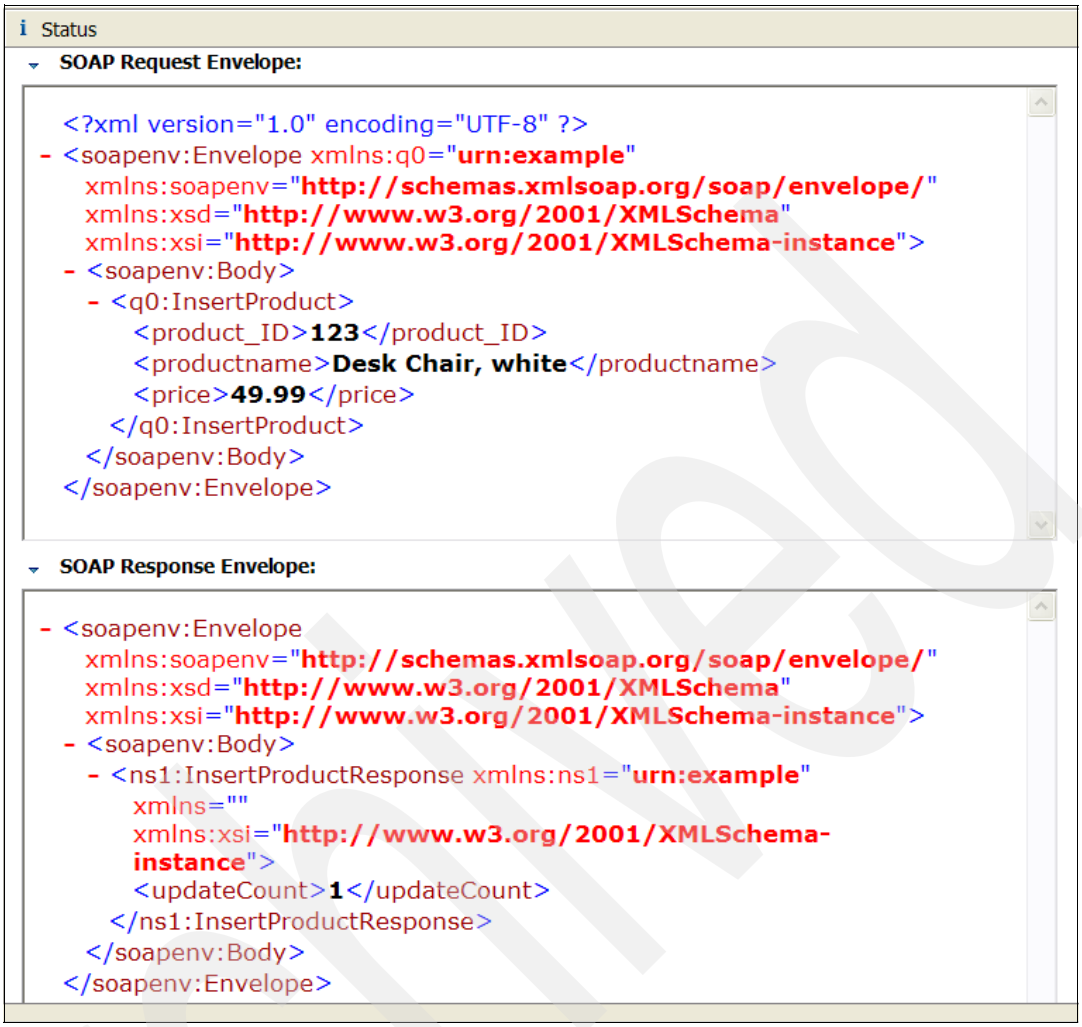


Figure 47 Status source view of InsertProduct

Testing the SelProductById component

Now that you have inserted your row into the PRODUCT table, you can use a second component of your Web service, the SelProductById component:

1. On the left side of the Web Services Explorer, click **SelProductById**.
2. On the right side of the Action pane, you are now asked to enter a PRODUCT_ID. Enter the product ID that you just used for your insert, and click **GO**.

   Figure 48 on page 47 shows you the results of the Status pane of the Web Services Explorer as a Form view.

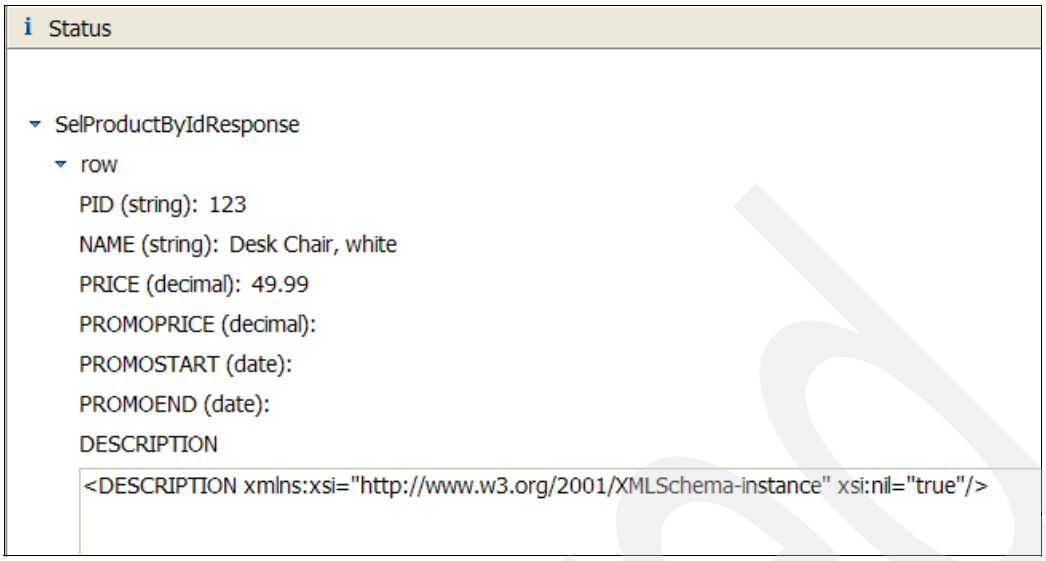


Figure 48 Status form after the execution of the SelProductById component

If you compare the result with the column names of the Product table, as shown in Figure 15 on page 18, you can see that those results are returned after you invoke the SelProductById component of your Web service MyFirstWebServiceSOAP.

Every message that is sent to and from a Web service is in XML format. The result, in Figure 48, does not really give you any indication that these messages are in XML format, because the Data Studio Web Services Explorer converted the SOAP message, which was returned in XML format, to a readable format for you. As opposed to the formatted result in Figure 48, Figure 49 on page 48 gives you the source view of the SOAP request and response messages.

In the upper part of Figure 49 on page 48, you see the *SOAP request Envelope*. The lower part shows the *SOAP response Envelope*. We cut off both parts a little, but in the response Envelope, you see the complete soapenv:body part, which contains the actual result for the executed SelProductById component of your Web service.

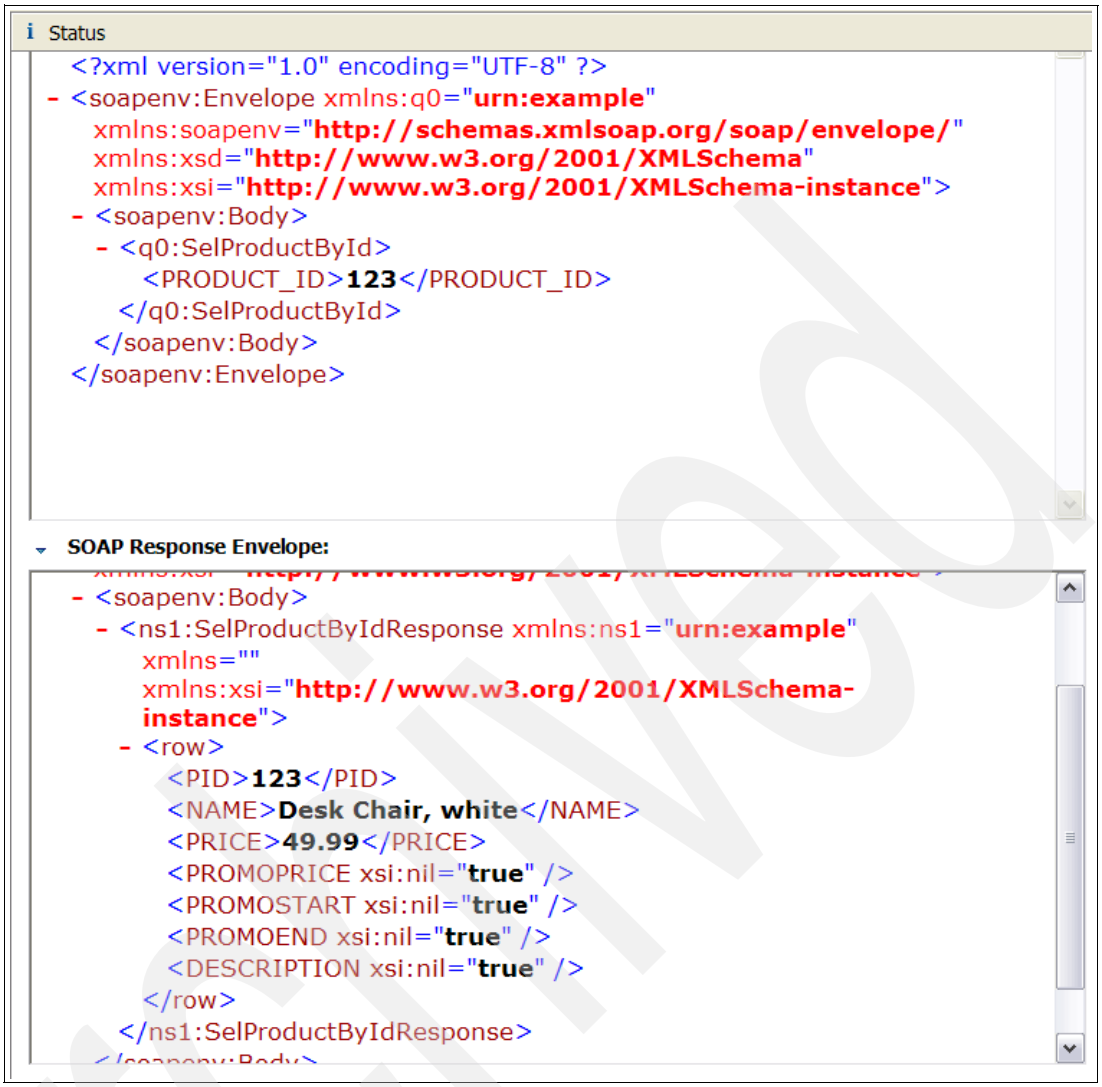


Figure 49 Status source after execution of select component

With what you have learned so far regarding the InsertProduct and SelProductById components of your Web service, you can now also test the other three components using the Web Services Explorer.

The Web Services Explorer is one handy choice to get started with Web services and to test them. If you are the Web service provider, you are only responsible for the implementation and the supply of Web services, so you might be finished at this point.

However, you might not only want to provide Web services to other individuals, but maybe also use Web services that are provided by other individuals. If you are interested to see how you can invoke your recently built and deployed Web service as a Web service consumer, we also show you a simple way to invoke it from DB2 and store, for example, the results of your SelProductById component in another table on your DB2 for z/OS subsystem. In “Invoking the Web service with SOAP UDFs” on page 53, we describe the consumer part of our example.

For now, we encourage you to continue reading the next section where we complete our testing.

Using the Test Client

Besides launching the Web Service Explorer for service testing, you can also add a Web Test Client to the generated service artifacts. This client is HTML-based and can be invoked from any Web browser without needing to have Data Studio running.

Note: The HTML-based Test Client is a priced feature and requires the Data Studio license if you use it after the 30 day trial period.

Generating the Test Client artifacts

In order to have Data Studio generate the Test Client artifacts, you must go to your Data Project Explorer and start the build and deploy for your Web service. If you do not remember how to do it, just go back to Figure 29 on page 30. As part of the build and deploy process, you see the Deploy Web Service build and deploy window appear, as shown in Figure 50.

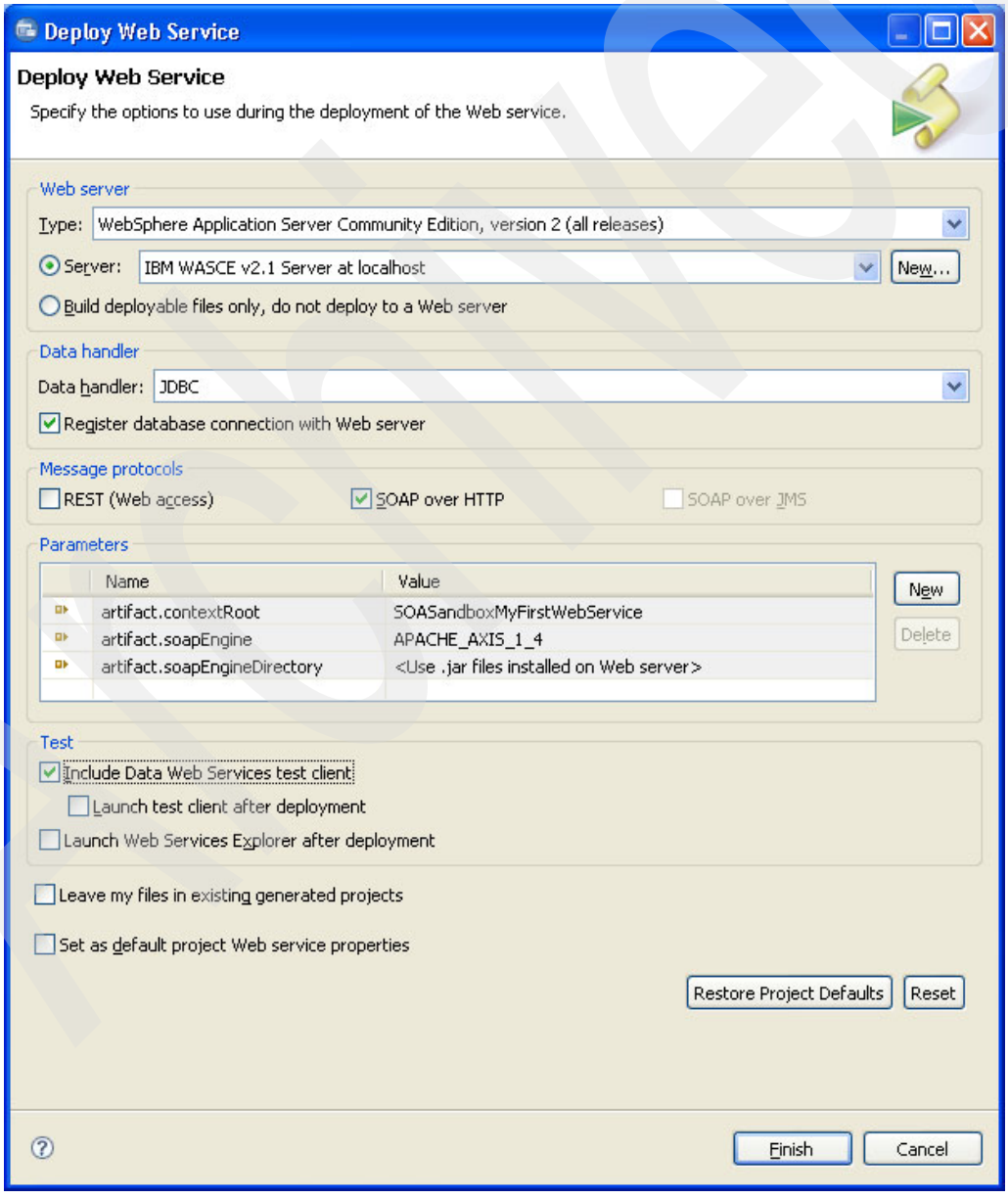
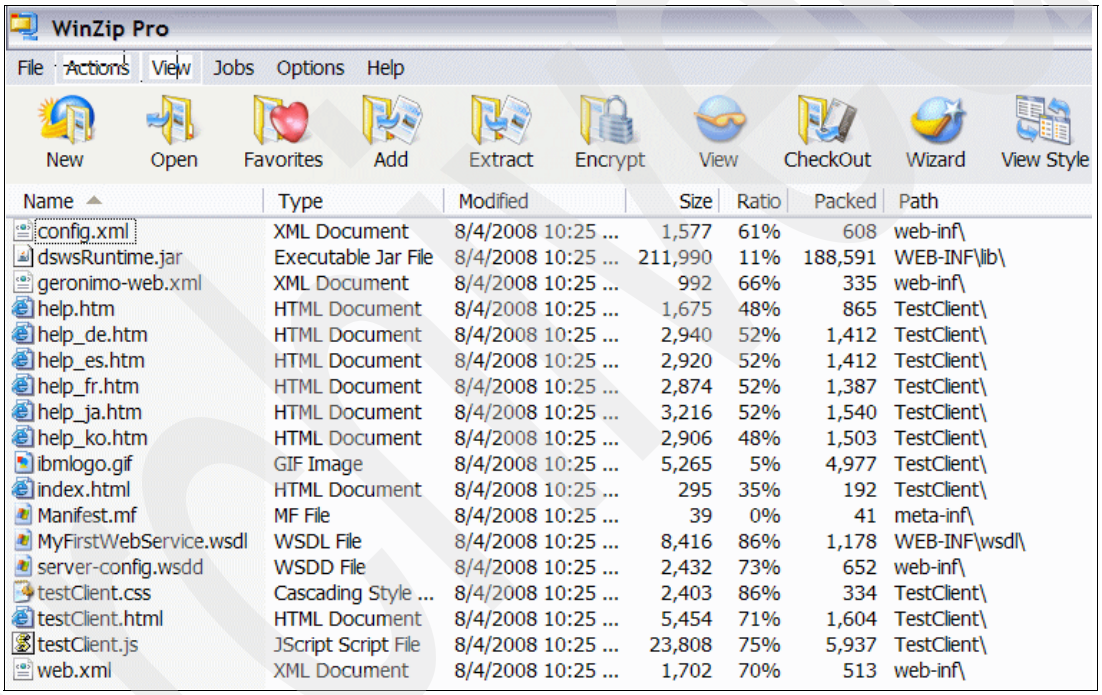


Figure 50 Build and deploy Web Service including the Test Client

The only two selections that we have added here under Test, as opposed to the previous build and deploy, are that we now ask Data Studio to include the artifacts for the Test Client by clicking **Include Data Web Services test client**. Because the Test Client is another way of testing your Web services, you must also select **Launch Web Services Explorer after deployment**. You might select the “Launch test client after deployment” option instead, but when you use this option, the Test Client is started within Data Studio, which is similar to what you saw before when we used the Web Services Explorer. We think it is more useful to start the Test Client independently of Data Studio.

Click **Finish**. Data Studio generates your Web services artifacts and does not start anything upon completion. However, the generated .war file, which you can see in the Navigator view, now contains additional artifacts (that is, files) for the Test Client. Export the .war file to your workstation. You can refer back to the description for Figure 32 on page 34, if you cannot remember how to export the .war file to your workstation.

After you have opened the exported .war file, you see the files that are shown in Figure 51.



Name	Type	Modified	Size	Ratio	Packed	Path
config.xml	XML Document	8/4/2008 10:25 ...	1,577	61%	608	web-inf\
dswsRuntime.jar	Executable Jar File	8/4/2008 10:25 ...	211,990	11%	188,591	WEB-INF\lib\
geronimo-web.xml	XML Document	8/4/2008 10:25 ...	992	66%	335	web-inf\
help.htm	HTML Document	8/4/2008 10:25 ...	1,675	48%	865	TestClient\
help_de.htm	HTML Document	8/4/2008 10:25 ...	2,940	52%	1,412	TestClient\
help_es.htm	HTML Document	8/4/2008 10:25 ...	2,920	52%	1,412	TestClient\
help_fr.htm	HTML Document	8/4/2008 10:25 ...	2,874	52%	1,387	TestClient\
help_ja.htm	HTML Document	8/4/2008 10:25 ...	3,216	52%	1,540	TestClient\
help_ko.htm	HTML Document	8/4/2008 10:25 ...	2,906	48%	1,503	TestClient\
ibmlogo.gif	GIF Image	8/4/2008 10:25 ...	5,265	5%	4,977	TestClient\
index.html	HTML Document	8/4/2008 10:25 ...	295	35%	192	TestClient\
Manifest.mf	MF File	8/4/2008 10:25 ...	39	0%	41	meta-inf\
MyFirstWebService.wsdl	WSDL File	8/4/2008 10:25 ...	8,416	86%	1,178	WEB-INF\wsdl\
server-config.wsdd	WSDD File	8/4/2008 10:25 ...	2,432	73%	652	web-inf\
testClient.css	Cascading Style ...	8/4/2008 10:25 ...	2,403	86%	334	TestClient\
testClient.html	HTML Document	8/4/2008 10:25 ...	5,454	71%	1,604	TestClient\
testClient.js	JScript Script File	8/4/2008 10:25 ...	23,808	75%	5,937	TestClient\
web.xml	XML Document	8/4/2008 10:25 ...	1,702	70%	513	web-inf\

Figure 51 The .war file content, including the Test Client

Refer to “Data Studio Developer perspectives” on page 64 for the details of another way to explore the contents of the generated .war file.

Invoking the Test Client

To use the Test Client, open any Web browser. The generic name for the URL to use is:

```
http://server:port/ctxRoot
```

If you used Data Studio to generate your Web service, deployed your .war file to a J2EE server, and used the default settings, the context Root name is built as a Data Project concatenated with your Web service name, for example, `SOASandboxMyFirstWebService`, in case you also used the names that we used throughout this Redpaper.

For the server name, you can use the localhost of your real TCP IP address. Because your TCP IP address might change when you log off and on, we recommend that you use localhost.

WebSphere Application Server Community Edition is listening at port 8080, so you must specify 8080 as the port number.

In our sample, we entered the following URL in our Web browser:

```
http://localhost:8080/SOASandboxMyFirstWebService
```

The Test Client starts, as shown in Figure 52.

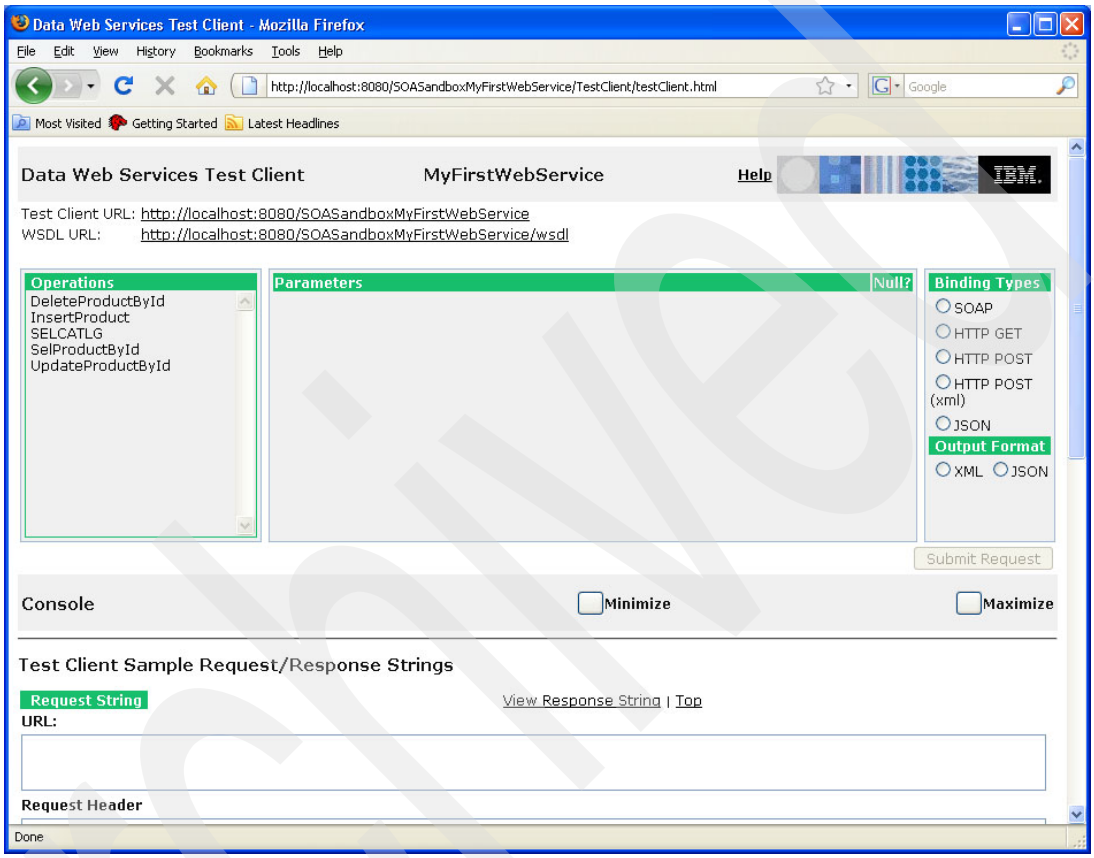


Figure 52 Test Client page

We describe the relevant information on Figure 52.

The upper part shows three boxes:

Operations Contains the components of your Web service. The four SQL scripts and stored procedure SELCATLG look familiar to you by now.

Parameters Used to enter input parameters for your Web service components.

Binding Types Prepares SOAP-type and REST-type message protocols using Data Studio. The various selections here represent the possible protocols, which are also called *bindings*.

On the left side, above the three boxes that we have just described, you see WSDL. If you click its URL, the Test Client returns your WSDL document.

Using the Test Client

The Test Client is a useful tool to:

- Test your Web services.

- Learn more about request strings. It is important to know what your request strings are as you move on to work with your first Web service as a client application.
- Learn more about response strings. The knowledge about how the Web service response is returned to you is very important as you move on, for example, calling your Web services through SOAP UDFs as shown in “Invoking the Web service with SOAP UDFs” on page 53.

To test the SelProductById component of your Web service:

1. In the Operations box, click **SelProductById**.
2. As you can see from Figure 53, you can now enter the input parameter for the SELECT statement in the middle box (the Parameters box.) Type 123 in here.
3. In the Binding Types area, select **SOAP**. Only SOAP binding is selectable in our case, because we only requested SOAP binding when we built and deployed the Web service using Data Studio as shown in Figure 50 on page 49.

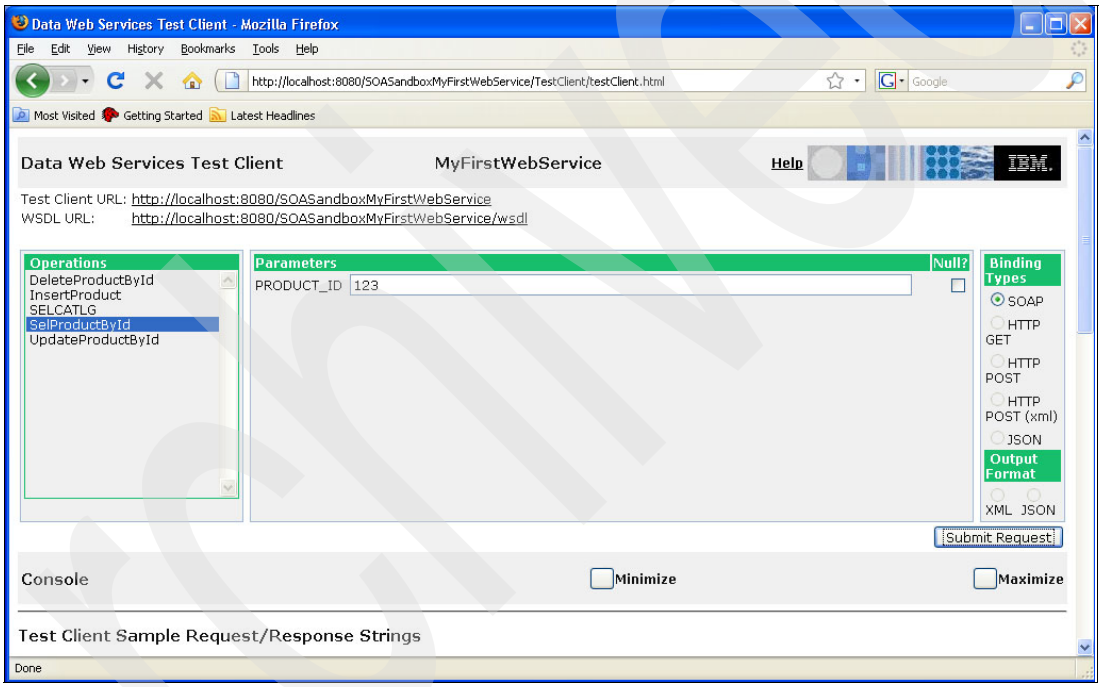


Figure 53 Execute SelProductById using Test Client

Executing the SelProductById component

To finally execute your Web service:

1. Click **Submit Request**, which is located on the right side, just below the Binding Types box.
2. To see the results, scroll down in your browser window. You can see two sections:
   - The request string
   - The response string

We explain the request string in the next section. Let us focus on the response string that is shown in Figure 54 on page 53. The upper box showing the Response Header is not important for us for now. The lower part of the Response Document is the interesting part. Embedded in the SOAP Envelope, you can see the real result of your SELECT statement. In the section in the large box with the yellow background, the actual result of the SELECT statement is contained in the row element.

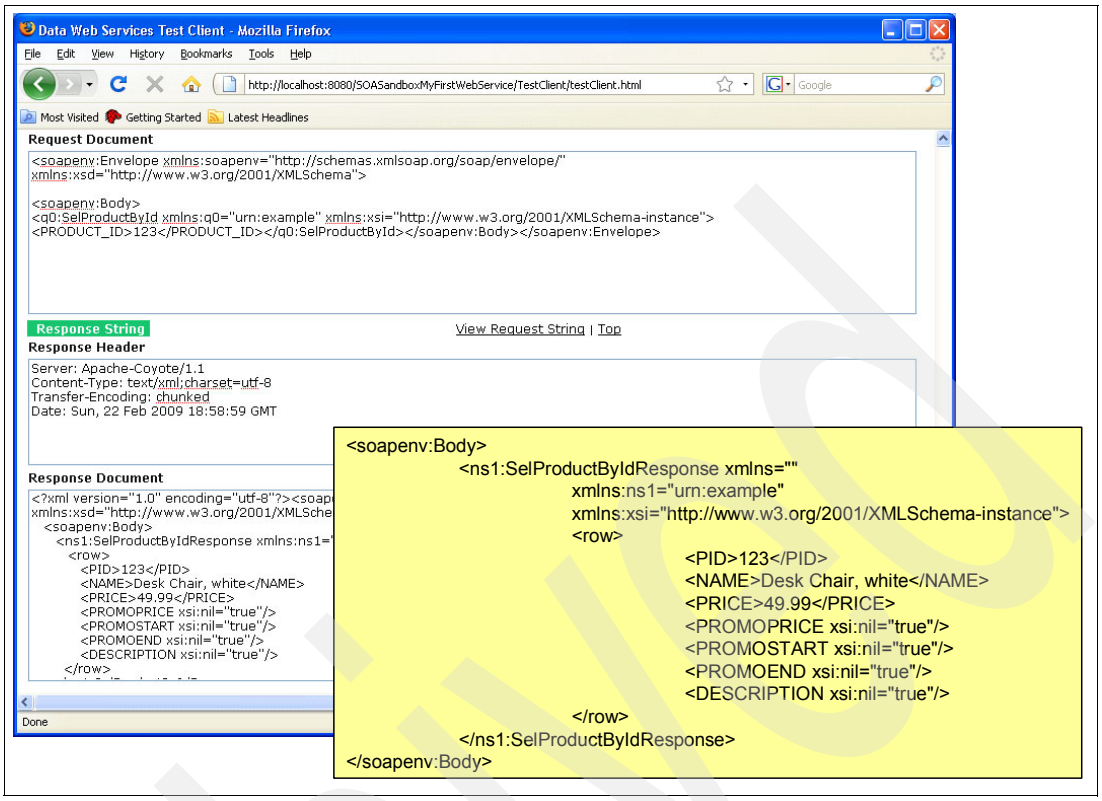


Figure 54 Response string

As seen before when we tested the SelProductById Web service, the result of this Web service is returned to you as a well-formed XML document.

If you are only interested in parts of the returned information, you must extract them using pureXML statements. However, you cannot use the test tools that we introduced, the Web Services Explorer that is built into Data Studio and the Test Client, for this kind of operation. In the next sections, we show you how you can only select the price of the product that you specify by product ID.

Invoking the Web service with SOAP UDFs

We have provided everything that you need to set up, build, deploy, and test the components that we have discussed in this document. For a possible extra step, invoking the SOAP UDFs from within an SQL statement, refer to Chapter 8. “DB2 as a Web services consumer with SOAP UDFs: Scenario 3”, of *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.

Invoking your Web service from SPUFI without XPath invocation

To get started, we show how you can simply invoke the Web service from SPUFI using a SOAP UDF. Figure 55 on page 54 gives you an overview of the sequence of events when you refer to one of the SOAP UDFs using a SELECT statement in SPUFI.

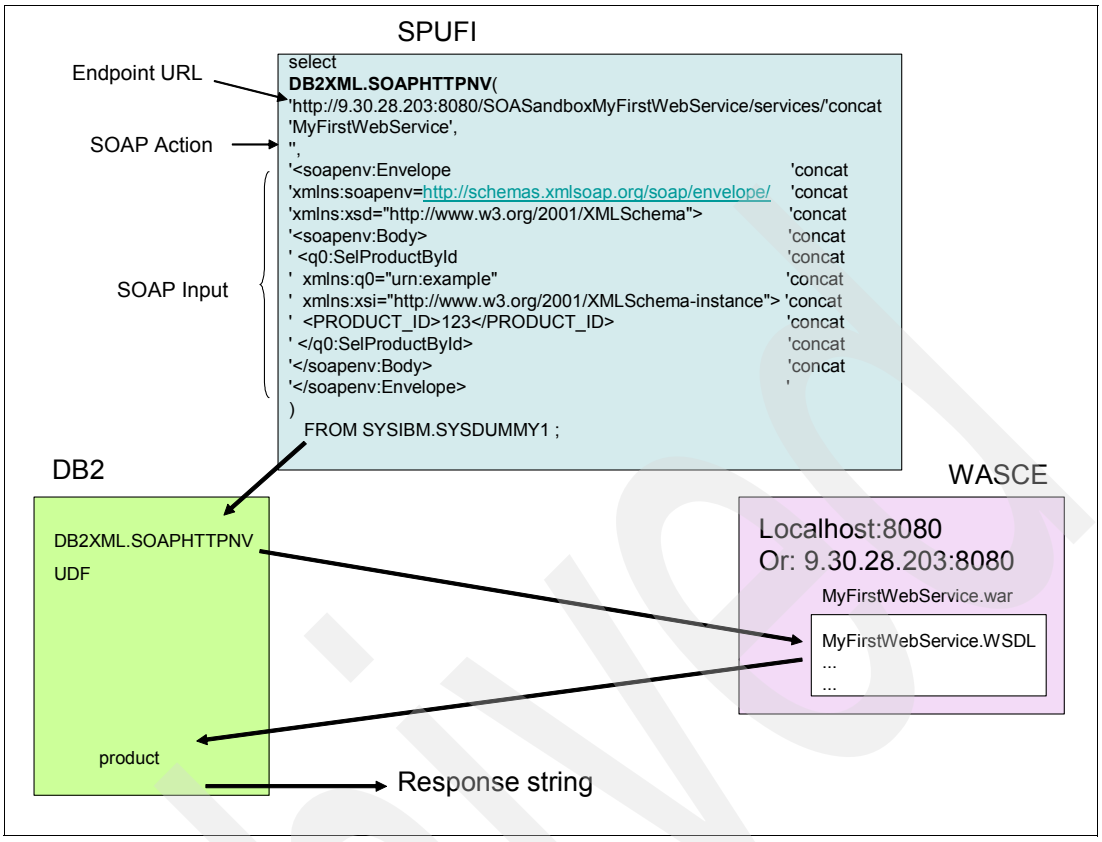


Figure 55 Invoke SOAP UDF from SPUFI

As with any UDFs, you can invoke the available SOAP UDFs using a SELECT statement, as shown in Figure 55. For the three input parameters to the SOAP UDFs, look at the Request string that is returned to you from the Test Client. Figure 56 on page 55 shows the Request string for the SelProductById component of your Web service. There, you can easily identify what you must use as input parameters for your UDF invocation:

Endpoint URL The URL shown in Figure 56 on page 55.

Here, you must change the IP address in the URL. The IP address must now reflect the URL on which WebSphere Application Server Community Edition is installed, which is your workstation IP address if you have followed our instructions so far. To obtain its value, you can, for example, issue the command **`ipconfig`** on your DOS prompt.

SOAP Action You can omit this input parameter if you used Data Studio to build your Web service.

SOAP Input The complete Request Document, as shown in Figure 56 on page 55.

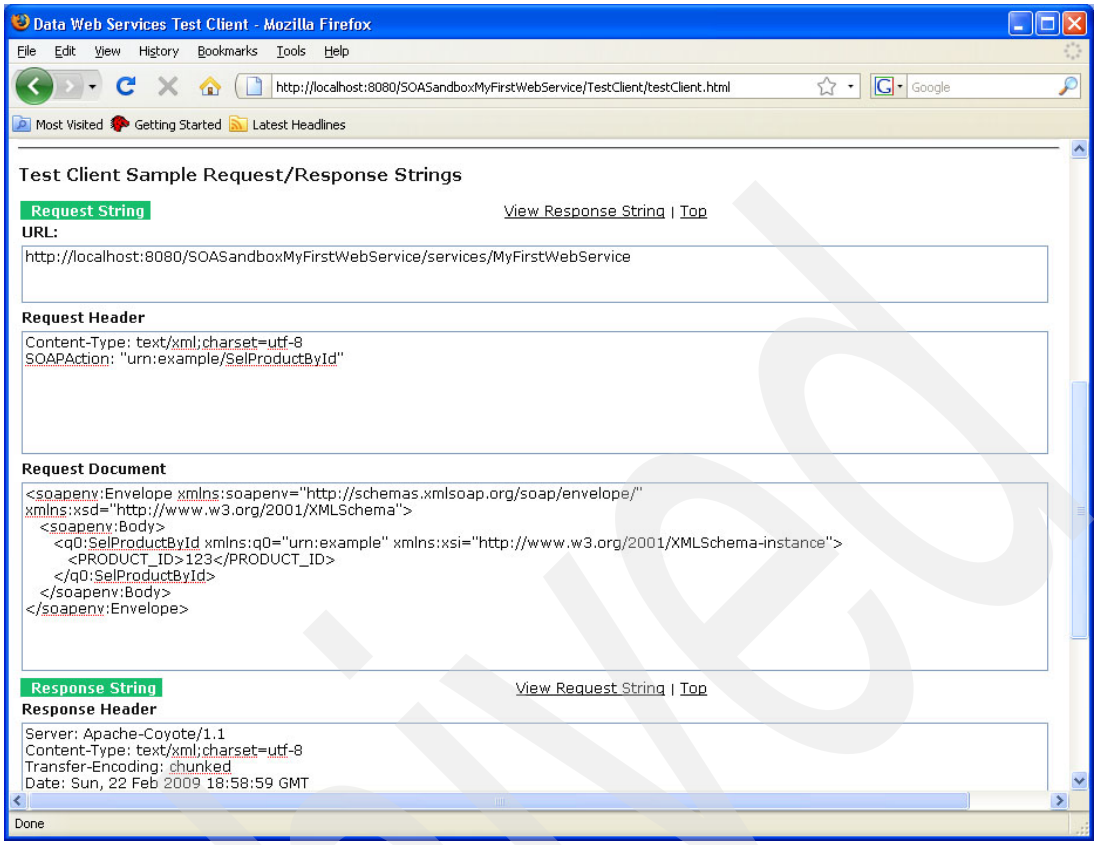


Figure 56 Test Client request string

When you issue the SQL statement, as shown on Figure 55 on page 54, the response that you receive is the same as the response that was shown in the Response string as Response Document. The only difference is that within SPUFI, it shows as a hard-to-read, long text string.

Invoking your Web service from SPUFI with XPath invocation

Let us now assume that you are not interested in the whole result row, but that you only want to know the price for the specific product for which you queried.

As we mentioned in “IBM Data Studio” on page 2, pureXML was introduced in DB2 9 for z/OS. The way to select specific information from well-formed XML documents is to use XPath functions. Figure 57 on page 56 shows the same UDF invocation that we used before, but in this case, we use XPath to select the price for product ID 123.

```
select xmlserialize( xmlquery('//PRICE/text()' passing (
xmlparse(document
(select DB2XML.SOAPHTTPNV(
'http://9.30.28.203:8080/SOASandboxMyFirstWebService/services/'concat
'MyFirstWebService',
'',
'<soapenv:Envelope                                                    'concat
'xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"            'concat
'xmlns:xsd="http://www.w3.org/2001/XMLSchema">                        'concat
'<soapenv:Body>                                                       'concat
' <q0:SelProductById                                                  'concat
'  xmlns:q0="urn:example"                                            'concat
'  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">            'concat
'  <PRODUCT_ID>123</PRODUCT_ID>                                      'concat
' </q0:SelProductById>                                               'concat
'</soapenv:Body>                                                    'concat
'</soapenv:Envelope>                                                '
)
   FROM SYSIBM.SYSDUMMY1 )))) as clob(1k))
   from sysibm.sysdummy1
```

Figure 57 XPath usage with SOAP UDF invocation

As a result in SPUFI, you get back one row that shows `49.99`.

What stylesheets can do for you

Extensible Stylesheet Language Transformation (XSLT) can transform an XML document's source tree into a different result tree. The XSLT language, which is a W3C specification, contains a set of rules describing those transformations.

To give you a hands-on feeling about what those stylesheets are and what they can do for you, we show you an easy way to transform your Web service response into a service message that is written in HTML format. All Web browsers can interpret HTML code. As a result, you can invoke your Web service directly through a Web browser and receive a formatted response.

To follow the explanation in this section, take the following introductory steps:

1. Create a new SQL script that contains a SELECT statement on any table. We created SQL script `SELdept.sql`, which represents the SELECT statement in Example 2. DSN8910.DEPT is one of the sample tables that the DB2 system programmer usually creates during the installation and migration of a DB2 for z/OS subsystem. If you cannot remember how to create a new SQL script, you can refer back to "Creating SQL scripts" on page 17.

Example 2 SELECT FROM DEPT

```
SELECT *
FROM DSN8910.DEPT
```

2. Create a new Web service. We created the new Web service `LearnAboutXSLT`. If you cannot remember how to create a new Web service, refer back to "1- Defining a new Web service in your Data Project Explorer" on page 25.
3. Drag-and-drop your SQL script to your new Web service.

Applying XSLT to your Web service operation

After you complete the three basic preparation steps for a new Web service, you are ready to learn how to use Data Studio Developer to transform the Web service response into a readable HTML format.

Creating a stylesheet

You must create a stylesheet document. Because you might not be familiar with XSLT, we created a stylesheet document and visualized it in Figure 58.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt">
    <xsl:output method="html" encoding="UTF-8" media-type="text/html"/>

    <xsl:template match="/*">
    <html>
          <head>
           <title>All departments</title>
          </head>
          <body>
            <table border="1">
               <tr bgcolor="#9acd32">
                 <xsl:if test="//row">
                    <xsl:for-each select="//row[1]/*">
                       <td style="width:150px"> <b><xsl:value-of select="local-name()"/></b></td>
                    </xsl:for-each>
                 </xsl:if>
               </tr>
               <xsl:for-each select="//row">
                  <tr>
                  <xsl:for-each select="*">
                       <td style="width:150px"> <xsl:value-of select="text()"/></td>
                    </xsl:for-each>
                  </tr>
               </xsl:for-each>
            </table>
          </body>
        </html>
    </xsl:template>
```

Title on Web browser

Assign column names to table

Width of result table columns in pixel

Assign contents to table cells

Figure 58 XSL document GeneralResponseAsHTML.XSL

The instructions in this stylesheet:

- Create a document that is encoded in unicode UTF-8 and that is built in HTML format.
- Give the Web page a name. In our specific case, the name is `All Departments`.
- Define the appearance of a table on your Web page: what the table border looks like, the color for it, the column width, and so on. We do not want to dive too deeply into the HTML language here, but we just want to give you an idea of what you can define for a table.
- Assign column names to the table.
- Assign contents to the table cells.

Our goal in this section is to make sure that you can follow the sample steps, so we keep steps simple for now and add complexity later. The stylesheet in Figure 58 is the one that we use later in this Redpaper. This stylesheet is a basic stylesheet, which can be used with any Web service response returning one or more rows as result of an SQL SELECT.

Assigning a stylesheet to a Web service operation

Now that you have a stylesheet that can be used to transform the XML response message of a Web service into an HTML document, you must connect it to the Web service operation that generates the response message:

1. On your new Web service, `LearnAboutXSLT`, click **SELdept**. You see the options that are shown on Figure 59.
2. From the pull-down menu, select **Manage XSLT**.

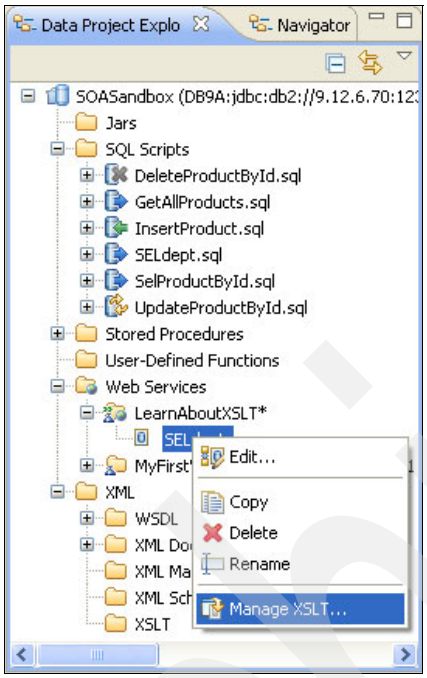


Figure 59 Add Manage XSLT to Web service operation

A new window opens. Refer to Figure 60 on page 59.

You have the option to transform both the Web service input and the output messages. In our example, we focus on just the output message.

3. To follow the sample, select **Browse** on the Output Messages section, navigate the browser, and select the path and file name that contain your .xsl document. Our XSL document is stored in z:\TEMP\RedPaper\GeneralResponseAsHTML.xsl.
4. After you select the correct file, click **Finish**. Your XSLT document is now associated to your Web service operation.

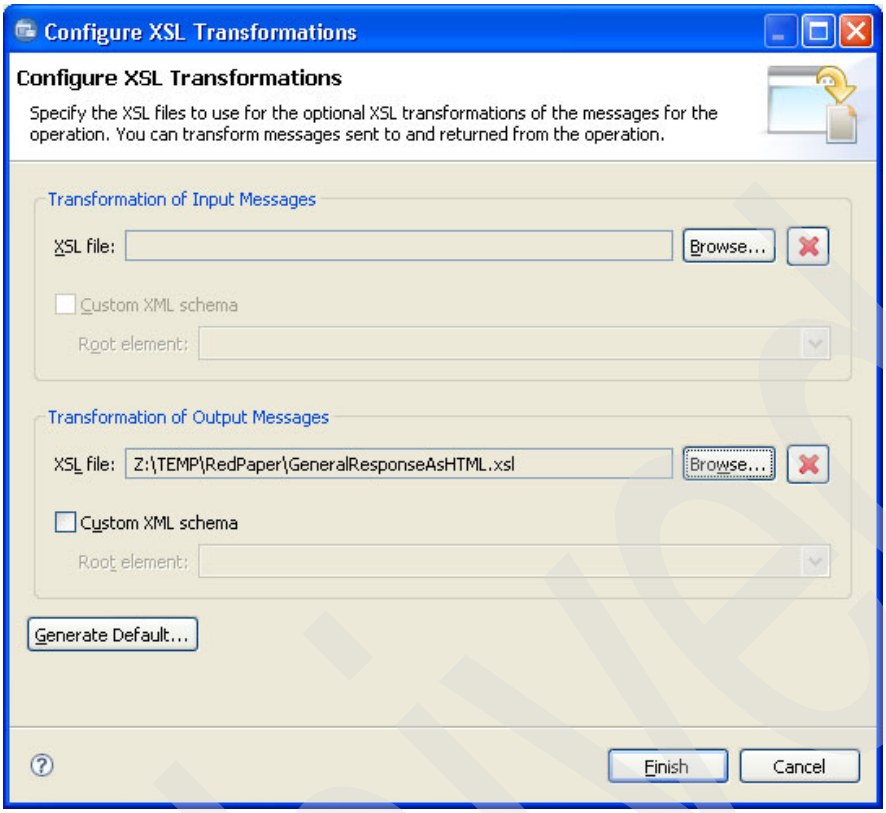


Figure 60 Add stylesheet to output messages

So far, you have only made associations, but you have not yet actually built and deployed your new Web service. To build and deploy your new Web service, in the Project Explorer view, click your Web service, and select **Build and Deploy**. The Build and Deploy Web Service window is displayed, as shown in Figure 61 on page 60.

During this Build and Deploy process, we select **REST** as the Message protocol, because Data Studio cannot apply the XSL transformation to SOAP output messages. If you select SOAP on the Build and Deploy windows, the operation does not appear in the SOAP binding of the service.

As for the other Build and Deploy processes previously seen, we choose to automatically deploy the Web service to our local J2EE server, which is the WebSphere Application Server Community Edition 2.1. In addition, we ask Data Studio to generate the Test Client that we introduced in “Using the Test Client” on page 49.

Click **Finish** to initiate the build and deploy.

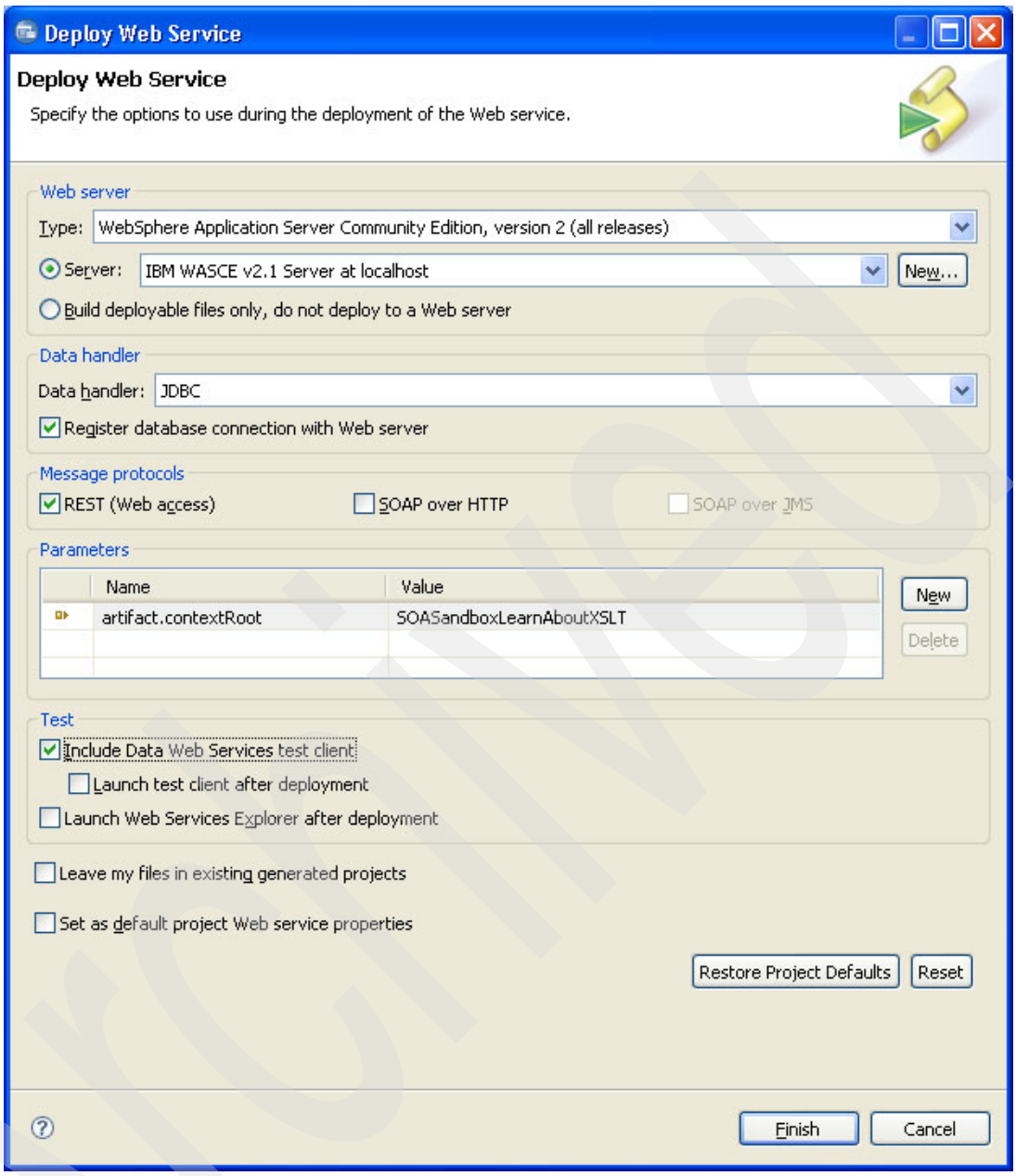


Figure 61 Build and Deploy Web service with XSL processing

Your Web service is now created and deployed to your WebSphere Application Server Community Edition server and therefore ready for use.

Testing the XSLT result

If you are not sure about how to invoke this REST Web service, you can get help from the Test Client that was generated upon your request. Invoking the Test Client is always the same and extremely simple. The format URL that you must use is:

```
http://webserver-IP:port/WebServicecontextRoot
```

In our case, this URL expands to:

```
http://localhost:8080/SOASandboxLearnAboutXSLT
```

If you use Data Studio Developer to create your Web services, the contextRoot is always a concatenation of your Data Project name followed by your Web service name.

Important: The address is case-sensitive. If you have issues when you try to invoke the Test Client, the first thing you might want to check is the lower and upper case definition.

To test the XSLT:

1. Open a Web browser, and enter the expanded URL. The Test Client starts for you.
2. On the upper left side of the browser window, select **SELdept Operation**. The window in Figure 62 is displayed.

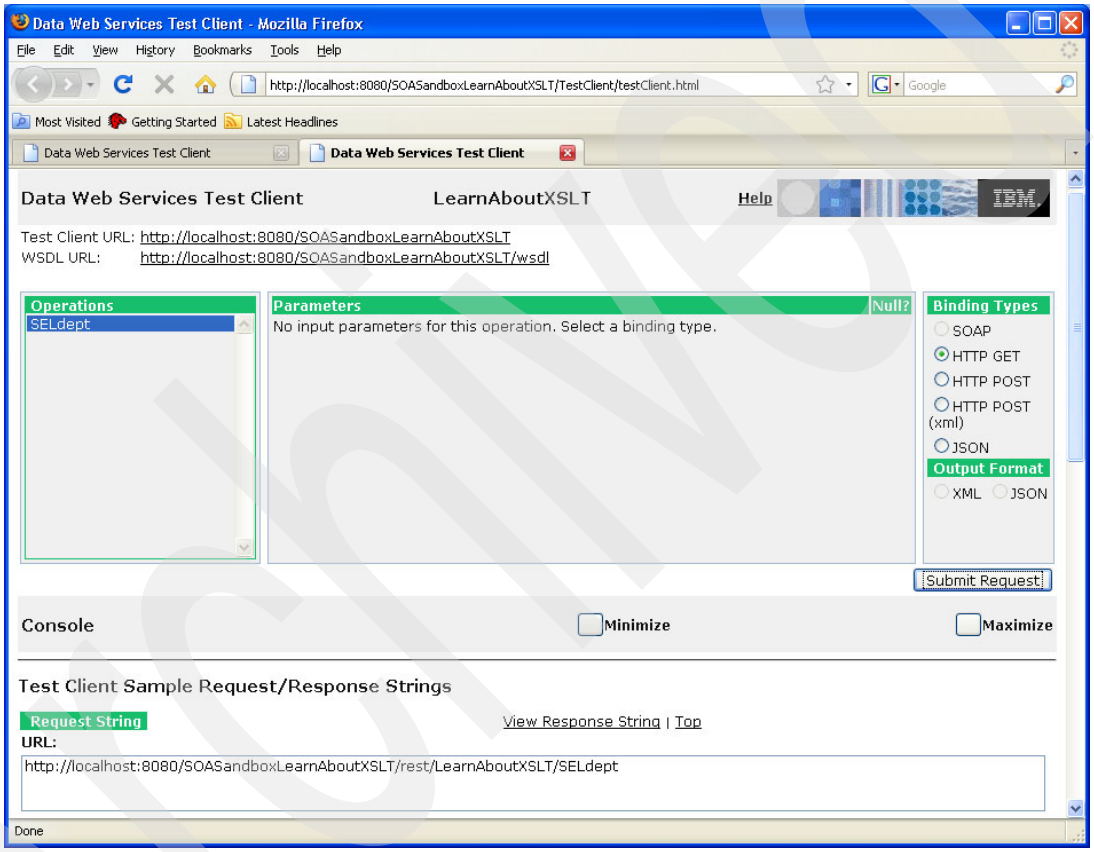
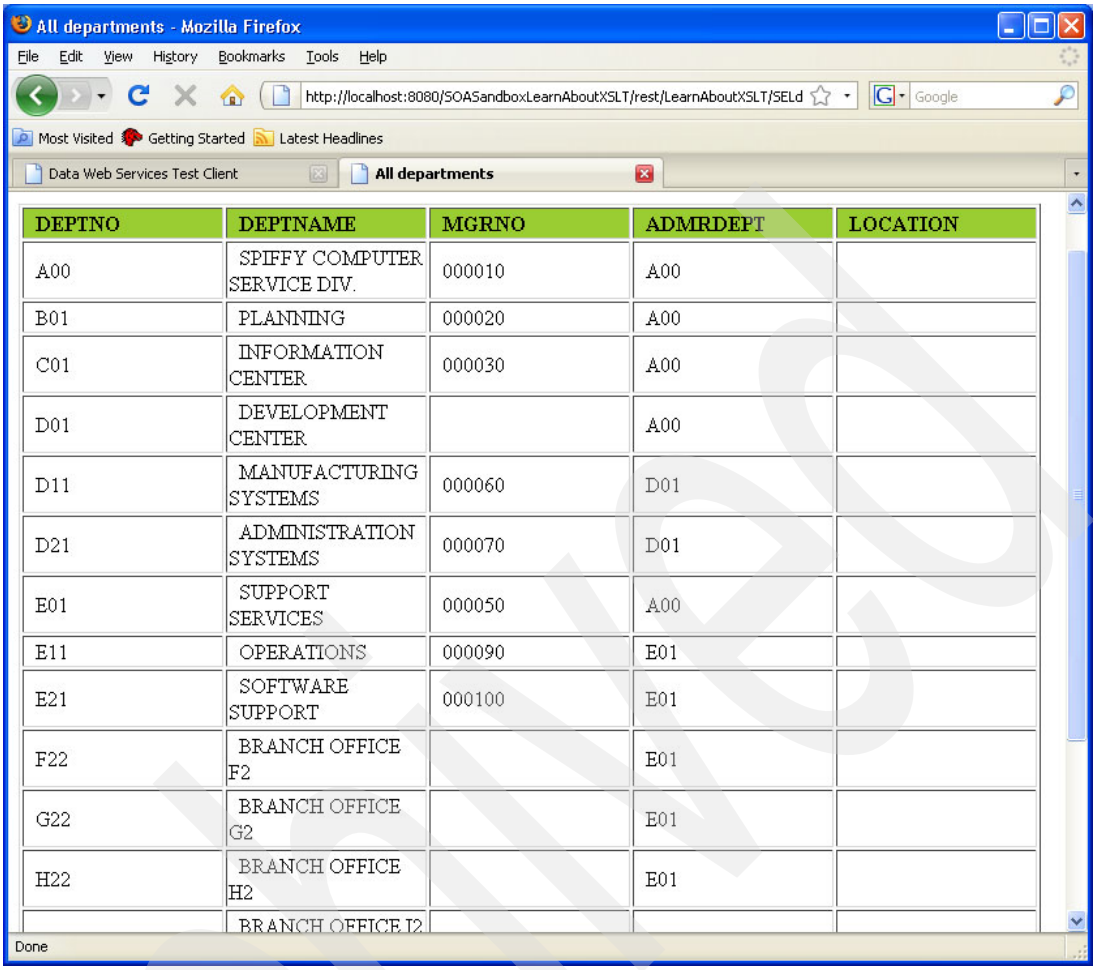


Figure 62 Test Client for LearnAboutXSLT Web service

3. Refer to the Request String in the lower part of Figure 62. Copy this request string and paste it into a Web browser address line to directly invoke your Web service. The response is displayed in Figure 63 on page 62.



DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENT CENTER		A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	
F22	BRANCH OFFICE F2		E01	
G22	BRANCH OFFICE G2		E01	
H22	BRANCH OFFICE H2		E01	
	BRANCH OFFICE I2			

Figure 63 Formatted HTML output for Web service operation SELdept

The output is nicely formatted and easy to read.

If you want to see the generated, transformed Web service response message as raw, unformatted data, you can either review the bottom part of the Test Client or right-click in the browser window, and select **View source** from the list of choices. When we did it this way, the result was Figure 64 on page 63. It is not necessary to see the complete result lines to get an idea of what the XSLT can do to format the message. So, we did not capture the complete document, which is longer and wider. We show just a portion of it.

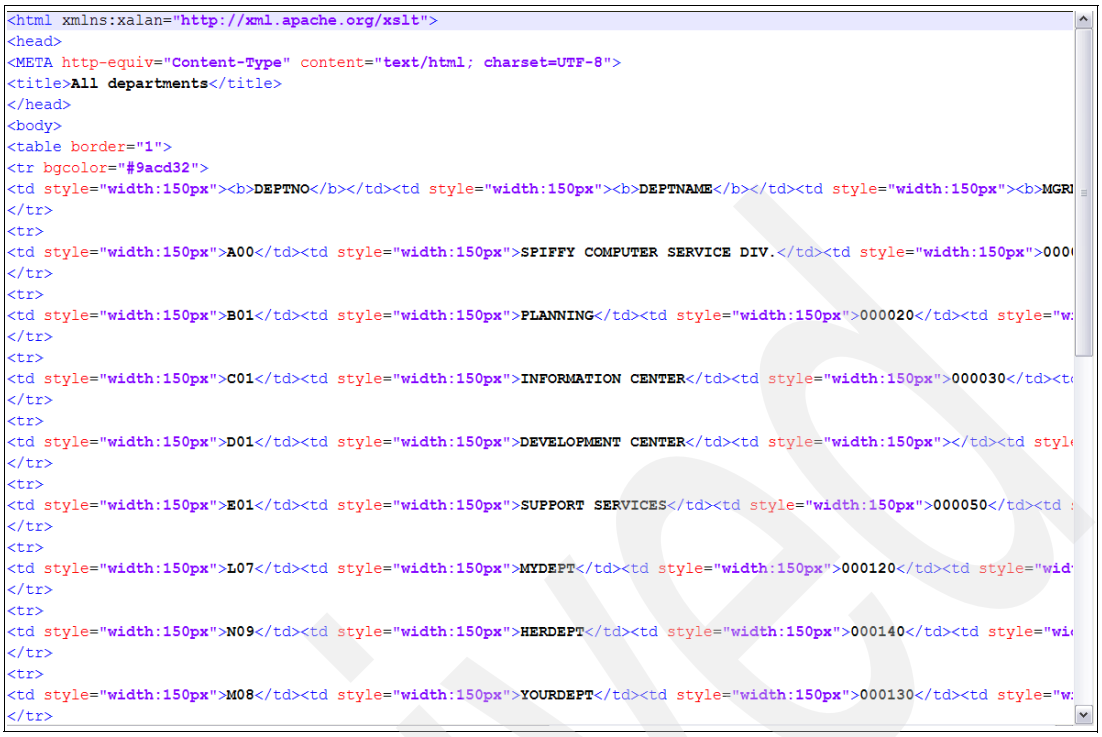


Figure 64 Source Web transformed response message

As you can see from Figure 64, the response message that is generated by the stylesheet attached to your Web service operation looks completely different from the response message without transformation that is partially captured for comparison in Figure 65. The information that we show is not complete, but absolutely sufficient for comparison.

<?xml version="1.0" encoding="UTF-8" ?>
<ns1:SELdeptResponse xmlns="" xmlns:ns1="urn:example" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
SERVICE DIV.</DEPTNAME><MGRNO>000010</MGRNO><ADMRDEPT>A00</ADMRDEPT><LOCATION>
</LOCATION></row><row><DEPTNO>B01</DEPTNO><DEPTNAME>PLANNING</DEPTNAME><MGRNO>0
</LOCATION></row><row><DEPTNO>C01</DEPTNO><DEPTNAME>INFORMATION CENTER</DEPTNAME
</LOCATION></row><row><DEPTNO>D01</DEPTNO><DEPTNAME>DEVELOPMENT CENTER</DEPTNAM
</LOCATION></row><row><DEPTNO>E01</DEPTNO><DEPTNAME>SUPPORT SERVICES</DEPTNAME><
</LOCATION></row><row><DEPTNO>L07</DEPTNO><DEPTNAME>MYDEPT</DEPTNAME><MGRNO>000
</LOCATION></row><row><DEPTNO>N09</DEPTNO><DEPTNAME>HERDEPT</DEPTNAME><MGRNO>0
</LOCATION></row><row><DEPTNO>M08</DEPTNO><DEPTNAME>YOURDEPT</DEPTNAME><MGRNO>
</LOCATION></row><row><DEPTNO>D11</DEPTNO><DEPTNAME>MANUFACTURING
SYSTEMS</DEPTNAME><MGRNO>000060</MGRNO><ADMRDEPT>D01</ADMRDEPT><LOCATION>
</LOCATION></row><row><DEPTNO>D21</DEPTNO><DEPTNAME>ADMINISTRATION SYSTEMS</DEPTN
</LOCATION></row><row><DEPTNO>E11</DEPTNO><DEPTNAME>OPERATIONS</DEPTNAME><MGRNO
</LOCATION></row><row><DEPTNO>E21</DEPTNO><DEPTNAME>SOFTWARE SUPPORT</DEPTNAME>
</LOCATION></row><row><DEPTNO>F22</DEPTNO><DEPTNAME>BRANCH OFFICE F2</DEPTNAME><M
</LOCATION></row><row><DEPTNO>G22</DEPTNO><DEPTNAME>BRANCH OFFICE G2</DEPTNAME><
</LOCATION></row><row><DEPTNO>H22</DEPTNO><DEPTNAME>BRANCH OFFICE H2</DEPTNAME><
</LOCATION></row><row><DEPTNO>I22</DEPTNO><DEPTNAME>BRANCH OFFICE I2</DEPTNAME><M
</LOCATION></row><row><DEPTNO>J22</DEPTNO><DEPTNAME>BRANCH OFFICE J2</DEPTNAME><M
</LOCATION></row><row><DEPTNO>K35</DEPTNO><DEPTNAME>JOG</DEPTNAME><MGRNO>000110
</LOCATION></row></ns1:SELdeptResponse>

Figure 65 SELdept response message XML document without XSL

Restriction: The general purpose XSLT that is shown in Figure 58 on page 57 does not work properly for columns that contain XML documents. Columns that contain an XML document just show blank values.

Hints and tips for Data Web Services

During this project, we extensively used Data Studio Developer as a tool for creating Web services. Data Studio Developer is a powerful tool that has a lot of functionality and usage options; a lot of them are obvious, and other options might need explanation. We put together a list of items that we think are useful. Most of them relate to the development or testing steps that we have described earlier. We listed them here for convenience.

Data Studio Developer perspectives

Each Workbench window contains one or more *perspective*s. A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources.

Data perspective

As an example, as shown in Figure 66, the Data perspective, by default, shows the Development Project Explorer on the upper left corner, the Data Source Explorer on the lower left corner, the Data Output on the lower right corner, and the Editor in the upper right corner.

Refer to the oval in the upper right corner. It currently shows Data as highlighted, which means that you are currently looking at and working with the Data perspective.

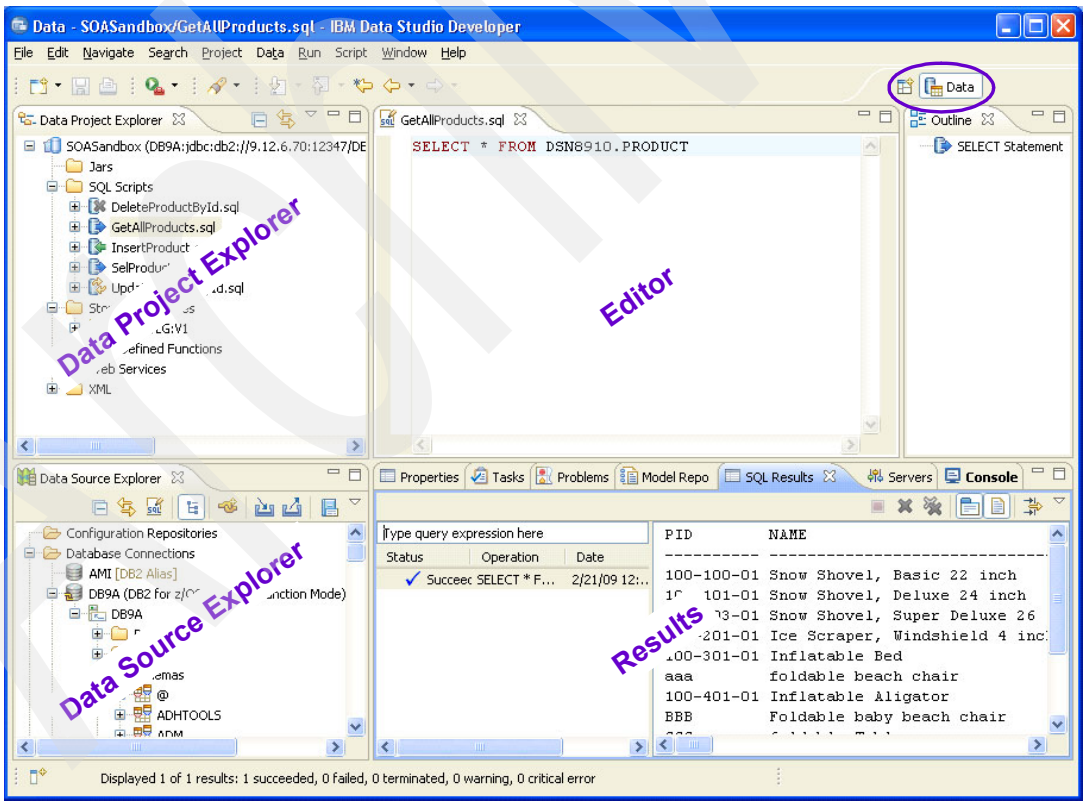


Figure 66 Data perspective

Perspectives control what appears in certain menus and toolbars. They define visible action sets, which you can change to customize a perspective. You can save a perspective that you have built in this manner, making your own custom perspective, to be opened again later.

Changing perspectives

Most of the functions that we use are invoked from the Data perspective, but we also use the J2EE perspective. Figure 67 gives you an overview of the available perspectives.

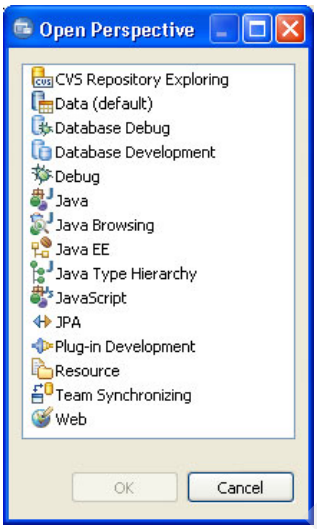


Figure 67 Available perspectives

J2EE is a perspective that you need when you work with Data Web Services. If you click **J2EE**, the perspective immediately changes, and the window in Figure 68 is displayed.

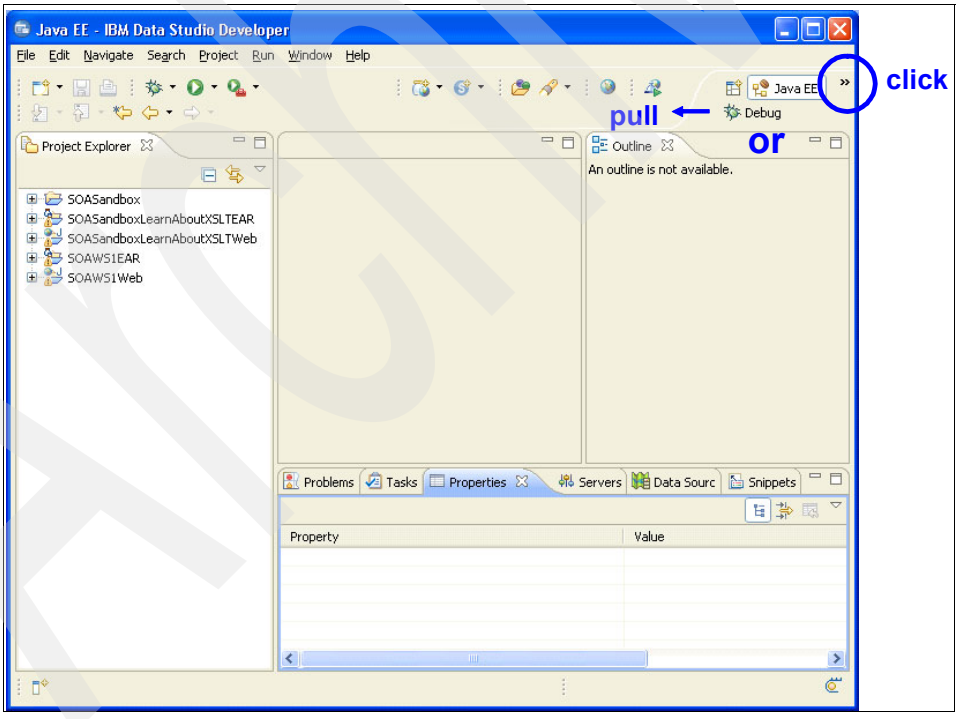


Figure 68 Perspective selection

If you want to switch back to the Data perspective, you can click the double-arrow on the right side, which shows you other already opened perspectives. Alternatively, move your mouse over the edge of the tab until a <-> is displayed. Move the whole tab to the left so that you see, for example, J2EE and Data as available perspectives.

J2EE perspective

Figure 68 on page 65 gives a general overview of the J2EE perspective. On the left side of this perspective, you see the Project Explorer view, which we explicitly captured in Figure 69.

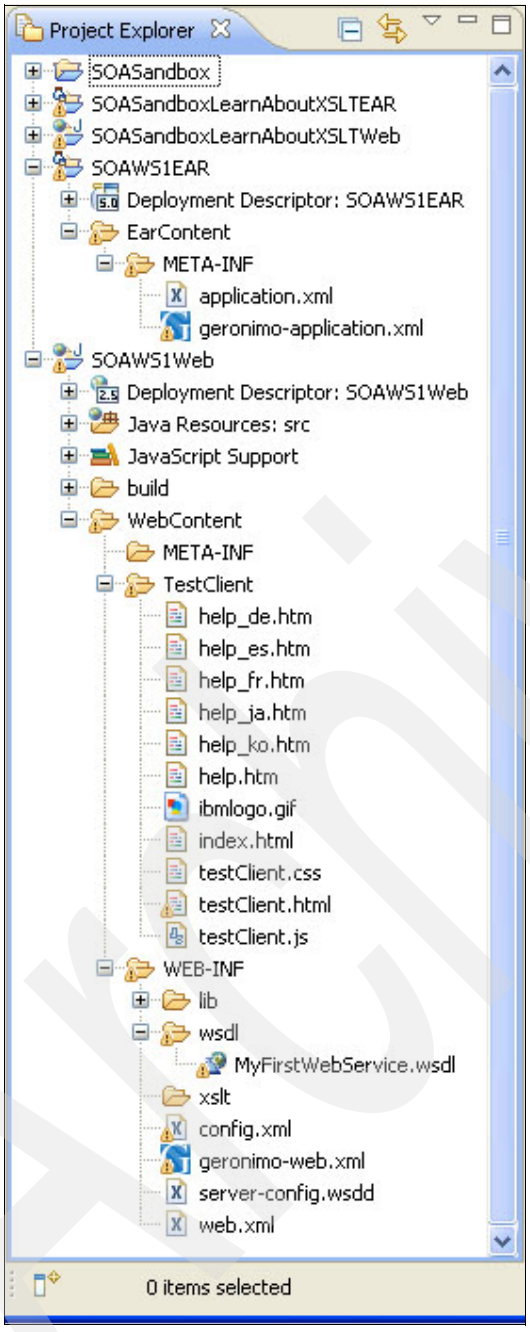


Figure 69 Project Explorer view in J2EE perspective

The two main expanded folders are:

- SOAWS1EAR
- SOAWS1Web

These two folders include the contents of the *.ear and *.war files. The WebContent folder contains all of the files and also shows the storage structure of the generated .war file. If you

are interested, you can compare the folder structure in here with what you see when you open a .war file using a compression tool.

Data Studio Developer properties

Data Studio is a powerful tool. Besides being able to perform many functions in a variety of ways using Data Studio Developer, you can also decide on different ways for Data Studio to perform its various tasks.

When you open the Preferences dialog box, you see an overview of the options for setting the Data Studio preferences. From the main menu, click **Window** → **Properties**. The Preferences dialog box is displayed, as shown in Figure 70.

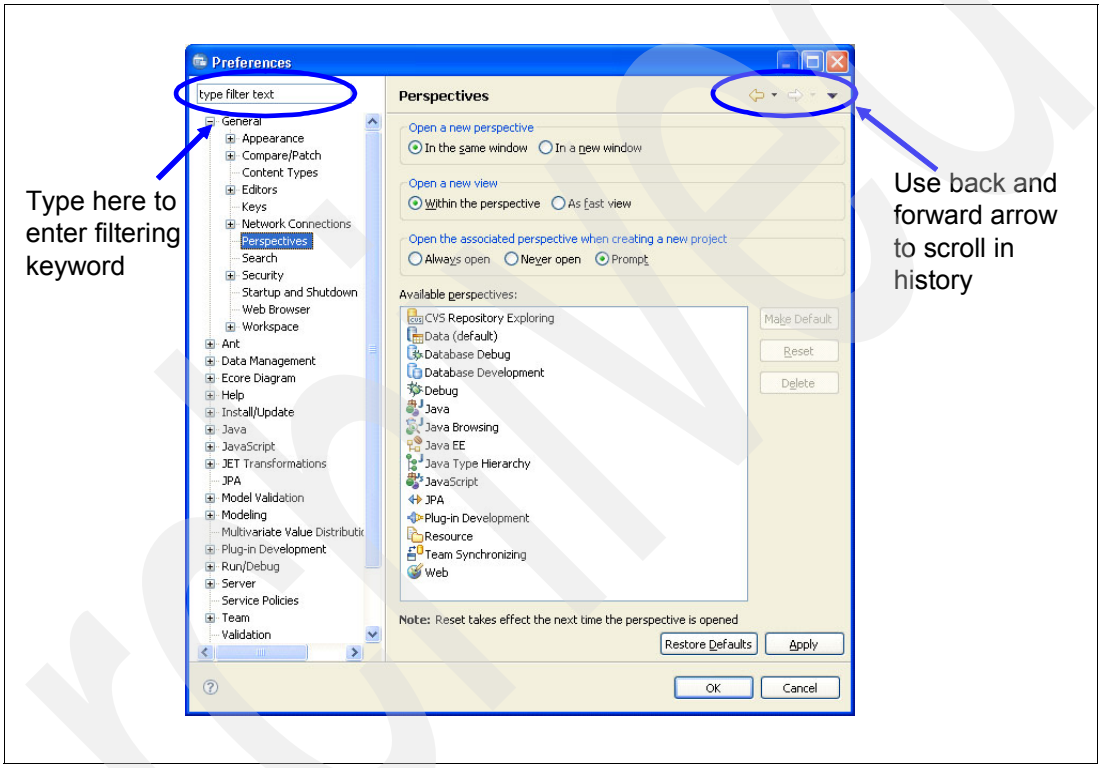


Figure 70 Preferences dialog box

The Preferences dialog is the dialog where you set user preferences. You can search the pages in the Preferences dialog using the filter function. To filter by matching the page title, type the name of the page that you are seeking into the input field on the upper corner and the available pages are presented below the input field. The filter also searches on keywords, such as appearance and Java.

Using the history controls, you can navigate through previously viewed pages. To step back or forward several pages at a time, click the drop-down arrow and a list of the most recently viewed preference pages is displayed. We do not list and discuss all the possible preference settings; we just point out that you have the flexibility to change them.

ContextRoot name

The contextRoot is the top-level directory of the application for the Web service, when the Web service is deployed to a Web server. We mentioned contextRoot earlier in “Testing the XSLT result” on page 60. Here, we add more information about contextRoot:

- The contextRoot is part of the URL that you use to invoke the Test Client. It is used as a third qualifier. The URL that you use typically looks like:

  ```
  http://webserver-IP:port/WebServicecontextRoot
  ```

- If you use Data Studio Developer to generate your Web services, the default contextRoot is always a concatenation of your Data Project name, followed by your Web service name.

  The result of this concatenation might be an undesirably long name. This contextRoot name is also used as the first part of the name of the .war file that is generated for you.

In Figure 71 on page 69, we ask for a build and deploy of the Web service, MyFirstWebService, which we developed in the Data Development Project SOASandbox. Following our previous statements, the names that Data Studio generates for you by default are:

- SOASandboxMyFirstWebService as contextRoot
- SOASandboxMyFirstWebServiceWeb.war for the .war zip file containing all the artifacts

Because the two names are cumbersome, in the Parameters section on the Deploy Web Service windows, we changed the parameter for the artifact.contextRoot to `SOAWS1`. As a consequence, Data Studio generates:

- SOAWS1 as contextRoot
- SOAWS1Web.war for the .war zip file that contains all of the artifacts

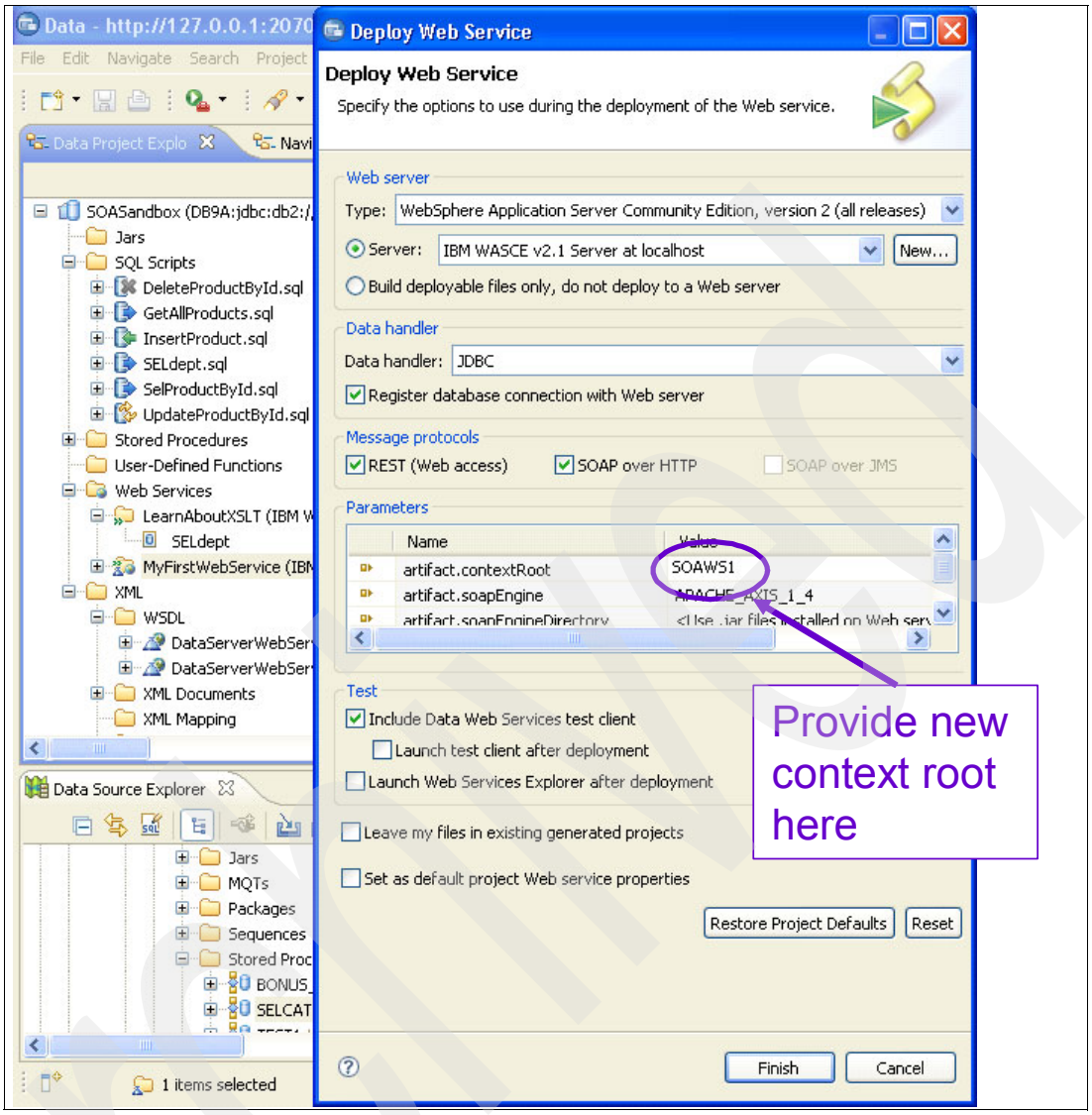


Figure 71 Change contextRoot name prior to Build and Deploy

In the Build and Deploy window shown in Figure 71, because you asked for the generation of the Test Client too, you can now invoke the Test Client using the following URL:

```
http://localhost:8080/SOAWS1
```

Testing with nullable input parameters

Refer back to our first Web service, which we introduced in “Creating your first Web service: Simple case” on page 25. One of the operations that we allocated to it was SQL script InsertProduct.sql. With the invocation of InsertProduct, you can insert one new row into table DSN8910.PRODUCT.

When we built our Web service, we used both SOAP and REST binding, but you will see that they have different behavior when you try to test your Web service using both bindings.

Nullable columns with SOAP binding

After successfully building and deploying your Web service, click your Web service, and from the pull-down menu, select **Launch Web Services Explorer**. The Web Services Explorer opens, and Figure 72 is displayed.

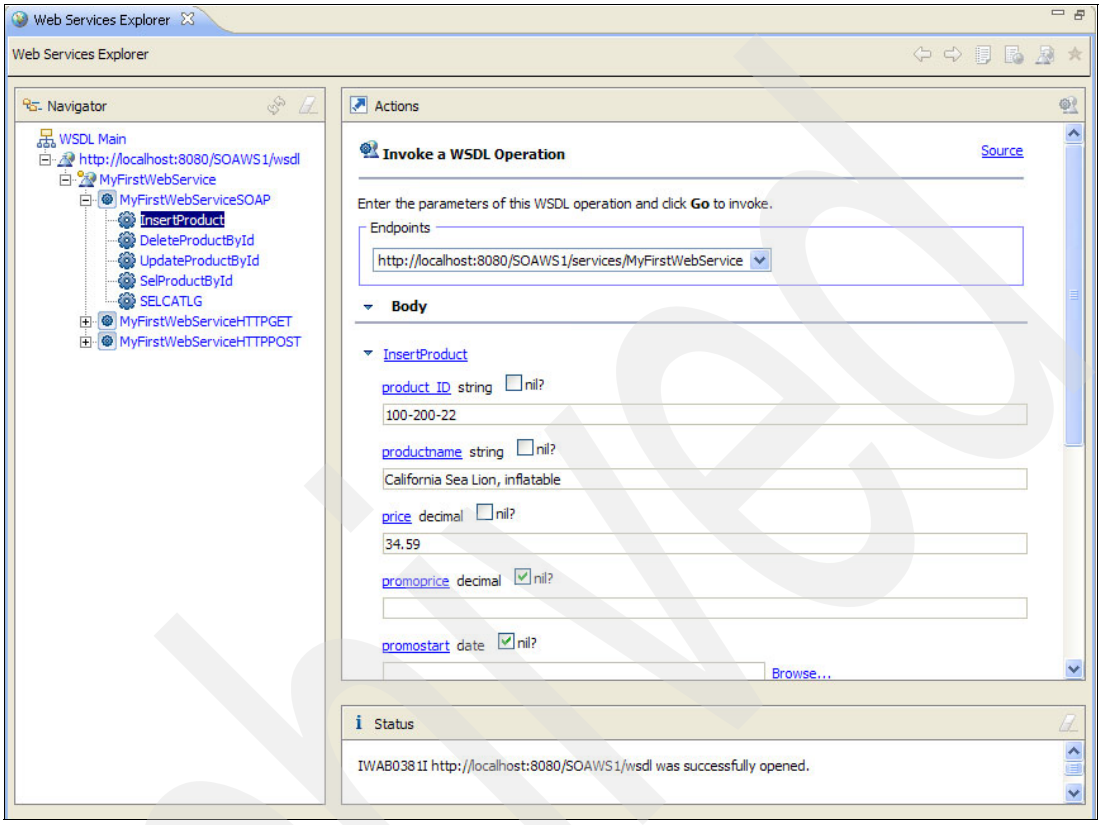


Figure 72 Web Services Explorer: SOAP and nullable columns

In the Actions pane, the Explorer displays all the input parameters. In our case, because we are attempting to insert a row, those are the table's column names. All columns are defined as nullable. So, if you, for example, intend to insert a new row for a Product that has a product ID, a productname, and a price, but no promoprice or promostart, you populate the parameters, as shown in Figure 72.

Note: We selected the nil? box next to the parameters promoprice and promostart to indicate that the input for these columns is NULL. We also selected the nil? box for the remaining columns, which you cannot see in Figure 72.

When you finally execute the Web service, the insert runs successfully and a new row is added to the table DSN8910.PRODUCT.

Nullable columns with REST binding

Apart from SOAP binding, we also asked for REST bindings. The REST bindings that were created for us are HTTP/POST and HTTP/GET, as you can see on the left side of Figure 73 on page 71. The Actions pane looks slightly different. One major difference is that there are no nil? boxes where you can specify that you are providing a NULL value for a specific column. Unfortunately, putting NULL into the parameter input string or leaving it blank does not work correctly.

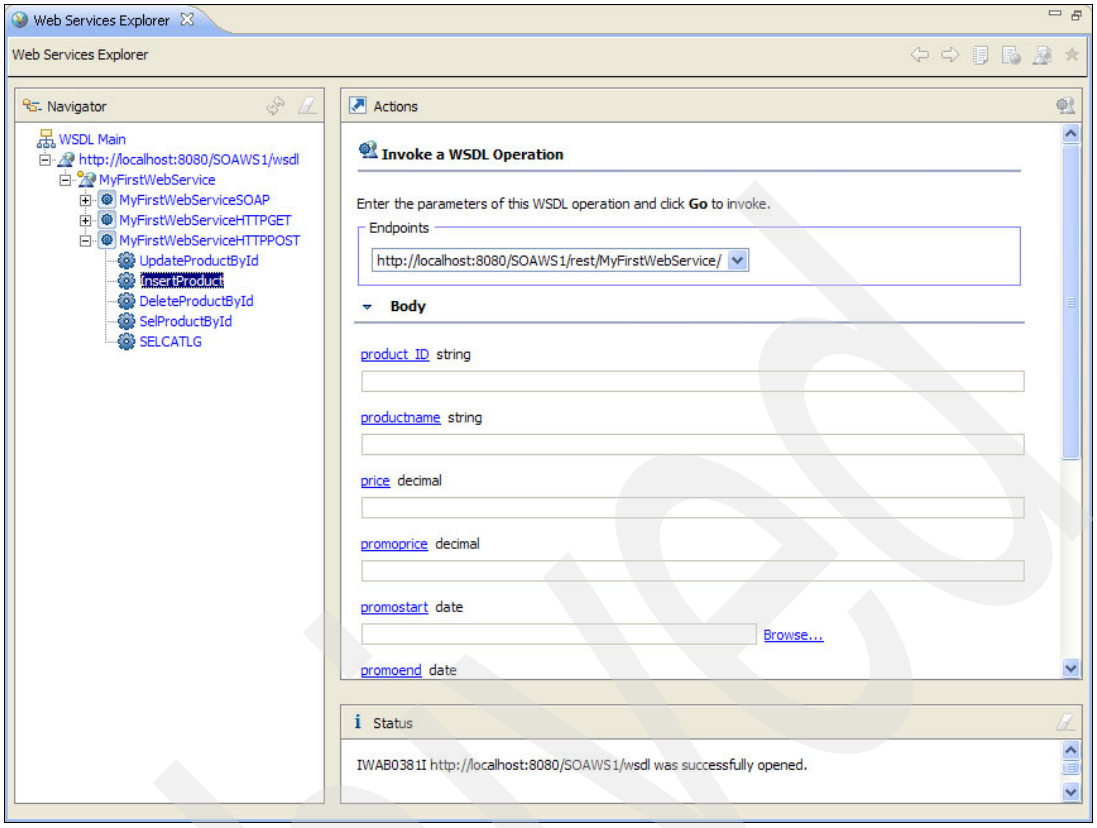


Figure 73 Web Services Explorer: REST and nullable columns

As a consequence, you cannot use the Web Services Explorer if you must specify a NULL value for a Web service that you want to test while working with REST binding.

Use the Test Client rather than the Web Services Explorer of Data Studio as a work-around if you need to specify NULL values using REST binding. Figure 74 shows the Test Client invoking the same Insert Operation using REST binding. The Test Client also gives you the capability to check the boxes under NULL? to indicate that a NULL value is supposed to be the input.

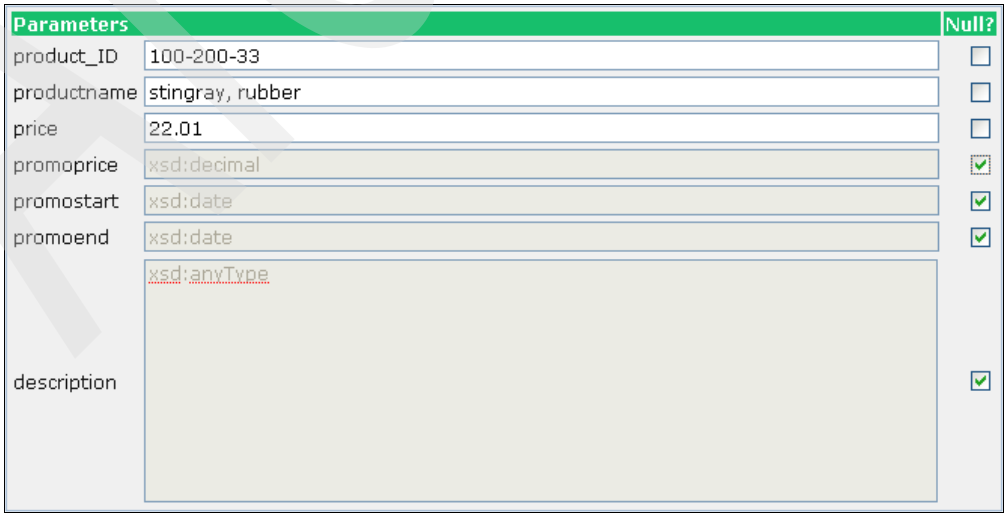


Figure 74 Use of Test Client to insert NULL values with REST

Resetting WebSphere Application Server Community Edition ports from Data Studio

When you try to start WebSphere Application Server Community Edition, you might encounter a problem that a server port is already in use. In this example, we assume the RMI Naming port is causing the problem, because it is already used by another program running on the system. To fix this situation, you have to change the RMI Naming port:

1. Go to the installation directory of WebSphere Application Server Community Edition.
2. Within this directory, follow path ..../var/config/config-substitutions.properties, and open the document. The document lists all of the ports that are used by WASCE.
3. Find the NamingPort entry and change the value, for example, from 1099 to 11099, as shown in Figure 75 on page 73.

```
# Put variables and their substitution values in this file.
# They will be used when processing the corresponding config.xml.
# Values in this file can be overridden by environment variables and system properties
# by prefixing the property name with 'org.apache.geronimo.config.substitution.'
# For example, an entry such as hostName=localhost
# can be overridden by an environment variable or system property
org.apache.geronimo.config.substitution.hostName=foo
# When running multiple instances of Geronimo choose a PortOffset value such that none
of the ports conflict.
# For example, try PortOffset=10
#Thu Nov 20 20:40:39 GMT+08:00 2008
AJPPort=8009
clusterNodeName=NODE
ORBPort=6882
MaxThreadPoolSize=500
ResourceBindingsNamePattern=
SMTPHost=localhost
ResourceBindingsQuery=?\#org.apache.geronimo.naming.ResourceSource
DerbyPort=1527
COSNamingPort=11050
webcontainer=TomcatWebContainer
OpenEJBPort=4201
ORBSSLPort=2001
PortOffset=0
ActiveMQStompPort=61613
JMXPort=9999
ORBHost=localhost
EndPointURI=http\://localhost\:8080
NamingPort=11099
DefaultWadiSweepInterval=36000
WebConnectorConTimeout=20000
HTTPSPort=8443
COSNamingHost=localhost
MinThreadPoolSize=200
ReplicaCount=2
ServerHostname=0.0.0.0
ActiveMQPort=61616
ORBSSLHost=localhost
SMTPPort=25
webcontainerName=tomcat6
ResourceBindingsNameInNamespace=jca\:
JMXSecurePort=9998
DefaultWadiNumPartitions=24
HTTPPort=8080
clusterName=CLUSTER_NAME
ClusterName=DEFAULT_CLUSTER
ResourceBindingsFormat={groupId}/{artifactId}/{j2eeType}/{name}
RemoteDeployHostname=localhost
TmId=71,84,77,73,68
```

Figure 75 The config-substitution.properties file

4. After you change the port number for your WebSphere Application Server Community Edition server, you must also propagate this information to your Data Studio Developer server entry:

   a. Double-click the WebSphere Application Server Community Edition server in Data Studio Developer, which opens a new view in the upper right corner of your window, as shown in Figure 76 on page 74.

b. In the upper right corner of the Server Configuration, there is an entry for the RMI port. Change the RMI Naming port to the number that you just chose for the config.xml file.

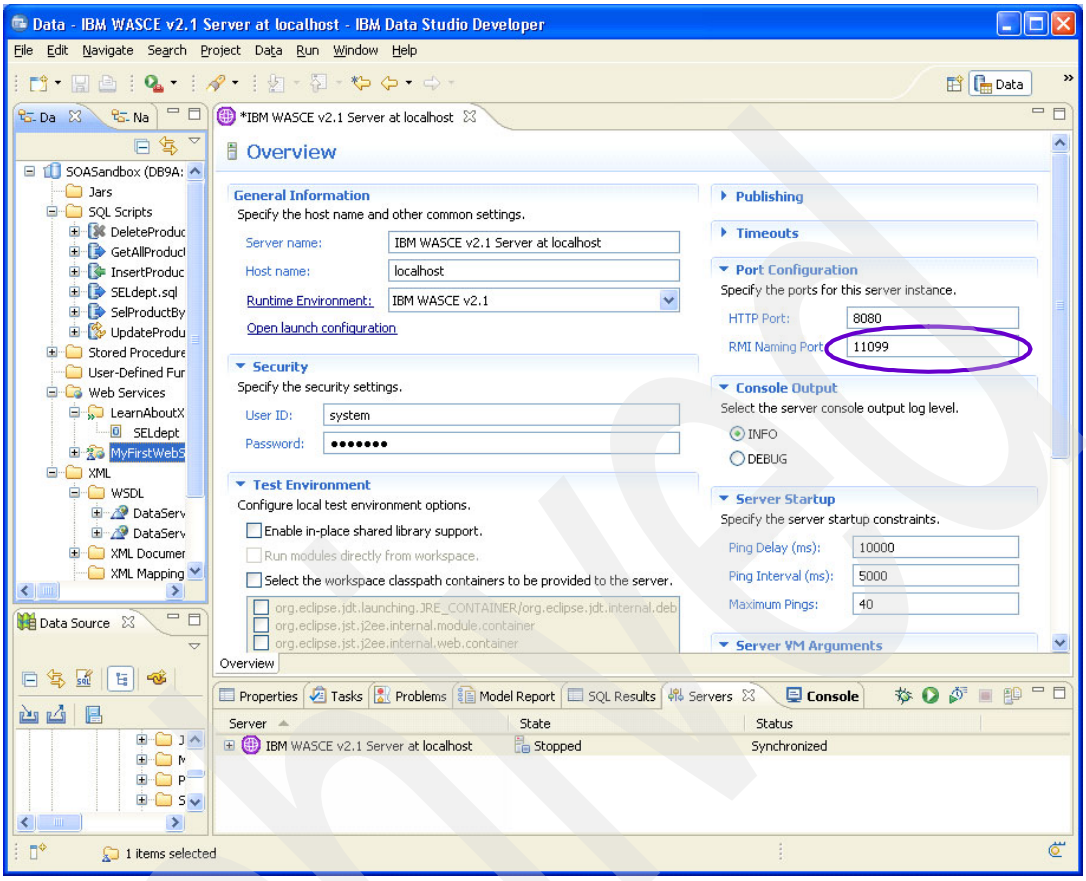


Figure 76 Change RMIport for the server definition

Enabling stored procedures for JDBC support

This appendix complements information about the Data Studio installation that is provided earlier in this document.

IBM Data Studio relies on the use of a set of DB2 stored procedures to extract information from the DB2 catalog. Normally, these stored procedures are installed by the member DSNTIJSG in your hlq.SDSNSAMP-partitioned data set during the DB2 installation process. If you did not create these stored procedures during DB2 installation, you must customize and run job DSNTIJMS to define the stored procedures that provide support for JDBC and ODBC client functions.

If these DB2 stored procedures are not defined or if the DB2 packages that are associated with these stored procedures are back-level, an SQL exception will result when you use IBM Data Studio, which indicates that the stored procedures are missing or that the required package is not found.

To determine if the required stored procedures were defined, issue the SQL statement in Example 3 on page 75 and ensure that the listed stored procedures are found. In our example, we used the default COLLID during the installation of these stored procedures.

Example 3 Checking the existence of collections for Java support

```
SELECT NAME,COLLID FROM SYSIBM.SYSROUTINES WHERE NAME LIKE 'SQL%'

NAME                 COLLID
-------------------- ------------
SQLCAMESSAGE
SQLCOLPRIVILEGES     DSNASPCC
SQLCOLUMNS           DSNASPCC
SQLFOREIGNKEYS       DSNASPCC
SQLGETTYPEINFO       DSNASPCC
SQLPRIMARYKEYS       DSNASPCC
SQLPROCEDURECOLS     DSNASPCC
SQLPROCEDURES        DSNASPCC
SQLSPECIALCOLUMNS    DSNASPCC
SQLSTATISTICS        DSNASPCC
SQLTABLEPRIVILEGES   DSNASPCC
SQLTABLES            DSNASPCC
SQLUDTS              DSNASPCC
```

If these stored procedures are not found, follow the execution instructions in the installation job DSNTIJSG or DSNTIJMS that is located in your hlq.SDSNSAMP to define the missing stored procedures.

If the stored procedures are found, the next step is to determine if the latest packages were bound, which is normally done as part of the installation process. If DB2 maintenance was applied to the system and one of the modules associated with the stored procedures listed in Example 3 was updated, perform a BIND to update the package that is associated to the new maintenance.

If this action is overlooked, a -805 SQLCODE is issued with the message text identifying the missing package name and time stamp, as shown in Example 4.

Example 4 SQL exception when using Data Studio

```
DEV7663.GETMRACT - Exception occurred while running:
A database manager error occurred.SQLCODE: -443, SQLSTATE: 38222 - ROUTINE SQLGETTYPEINFO
(SPECIFIC NAME SQLGETTYPEINFO) HAS RETURNED AN ERROR SQLSTATE WITH DIAGNOSTIC TEXT -805
DB9A.DSNASPCC.DSNATYP8.1856182D1761. SQLCODE=-443, SQLSTATE=38222, DRIVER=3.52.90
```

To correct this problem, modify the installation job DSNTIJSG or DSNTIJMS in your hlq.SDSNSAMP partitioned data set to BIND all the packages that are associated to the stored procedures in the previous listing. An easy way to identify the packages that are associated to the JDBC support stored procedures is to locate all of the BIND statements in DSNTIJSG or DSNTIJMS that have a reference to DSNASPCC, which is the default COLLID. You can extract and execute these BIND statements from DSNTIJSG or DSNTIJMS to bind just the packages that are associated to JDBC support stored procedures.

For more information about the JDBC support stored procedures, refer to “Enabling stored procedures and tables for JDBC and ODBC support” in the *DB2 Version 9.1 for z/OS Installation Guide,* SC18-9842.

Executing the job in Example 5 on page 76 might solve the issues that are related to missing binds for Data Studio packages. Update the default subsystem name (DSN), DB2 code library, and the database request module (DBRM) library to comply with your environment.

Example 5 Binding Data Studio-related packages

```
//PAOLOR5L JOB (999,POK),CSF-CKDS,
//         NOTIFY=&SYSUID,
//         CLASS=A,
//         MSGCLASS=X,
//         MSGLEVEL=(1,1),
//         REGION=0M
/*JOBPARM S=SC63,L=9999
//DSNTIRU EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB DD DSN=DB9A9.SDSNLOAD,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
   DSN SYSTEM(DB9A)
 BIND PACKAGE(DSNASPCC) MEMBER(DSNACPR8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNACOL8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNAFNK8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
      RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNAPRK8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNAPCO8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNAPRC8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNASPC8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNASTA8) -
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNATBP8) -
         ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
         RELEASE(COMMIT) ENCODING(EBCDIC) -
         LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNATBL8) -
         ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
         RELEASE(COMMIT) ENCODING(EBCDIC) -
         LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNATYP8) -
     ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  BIND PACKAGE(DSNASPCC) MEMBER(DSNAUDT8) -
```

```
       ACTION(REPLACE) ISOLATION(UR) QUALIFIER(SYSIBM) -
       RELEASE(COMMIT) ENCODING(EBCDIC) -
       LIBRARY('DB9A9.SDSNDBRM')
  END
/*
//SYSIN DD *
/*
```

As of December 2008, the tables in Example 6 have been added to the list of tables created by DSNTIJSG.

Example 6 Additional JDBC and SQLJ tables

```
SYSIBM.SQLTABLETYPES
SYSIBM.SQLTYPEINFO
SYSIBM.SQTCOLPRIVILEGES
SYSIBM.SQTCOLUMNS
SYSIBM.SQTCOLUMNS_OLEDB
SYSIBM.SQTFOREIGNKEYS
SYSIBM.SQTPRIMARYKEYS
SYSIBM.SQTPROCEDURECOLS
SYSIBM.SQTPROCEDURES
SYSIBM.SQTSPECIALCOLUMNS
SYSIBM.SQTSTATISTICS
SYSIBM.SQTTABLEPRIVILEGES
SYSIBM.SQTTABLES
SYSIBM.SQTUDTS
SYSIBM.UINDEXES
```

The SQL statements that generate these tables are in member DSNATMMC of data set "prefix.SDSNSAMP." You need to run DSNATMMC before the BIND job in Example 5 on page 76.

Previously, only SYSIBM.SYSDUMMY tables were listed as those tables created by DSNTIJSG. The SYSIBM.SYSDUMMY* tables are used by JDBC and ODBC to avoid unnecessary character conversion during the execution of functions, such as LEN() and SUBSTR(), when the arguments of these functions are CLOB or DBCLOB locators.

The other tables are used by the stored procedures for JDBC and ODBC support to store temporary copies of the result sets that the stored procedures return.

The team that wrote this IBM Redpaper

This Redpaper was produced by a team of specialists working at the Silicon Valley Lab, San Jose California.

Paolo Bruni is an ITSO Project Leader based in IBM Silicon Valley Laboratory. Paolo has authored several IBM Redbooks® publications and Redpapers on DB2 for z/OS.

Michael Schenker is a Software Engineer in IBM Information Management Group (part of IBM Software Group). He joined IBM in 2002. His specialities include XML, Web services and databases. Michael is the lead developer for Data Web Services functions of IBM Data Studio.

Special thanks to the following people

We want to acknowledge:

- Sabine Kaschta, who coauthored Chapter 4, “Getting started with Web services”, of *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663.
- Ernest Mancill, who helped in solving the cryptic issues of BIND.

References

For additional information:

- *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663-00
- *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond,* SG24-7604-00
- *DB2 Version 9.1 for z/OS Installation Guide,* SC18-9842-05

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-4510-00 was created or updated on April 6, 2009.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks