

SAP NetWeaver Gateway plug-in for Eclipse



Version 2.5.400



Copyright

© Copyright 2012 SAP AG. All rights reserved.

SAP Library document classification: PUBLIC

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, PowerPoint, Silverlight, and Visual Studio are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, z10, z/VM, z/OS, OS/390, zEnterprise, PowerVM, Power Architecture, Power Systems, POWER7, POWER6+, POWER6, POWER, PowerHA, pureScale, PowerPC, BladeCenter, System Storage, Storwize, XIV, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, AIX, Intelligent Miner, WebSphere, Tivoli, Informix, and Smarter Planet are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the United States and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are trademarks or registered trademarks of Adobe Systems Incorporated in the United States and other countries.

Oracle and Java are registered trademarks of Oracle and its affiliates.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems Inc.

HTML, XML, XHTML, and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Apple, App Store, iBooks, iPad, iPhone, iPhoto, iPod, iTunes, Multi-Touch, Objective-C, Retina, Safari, Siri, and Xcode are trademarks or registered trademarks of Apple Inc.

IOS is a registered trademark of Cisco Systems Inc.

RIM, BlackBerry, BBM, BlackBerry Curve, BlackBerry Bold, BlackBerry Pearl, BlackBerry Torch, BlackBerry Storm, BlackBerry Storm2, BlackBerry PlayBook, and BlackBerry App World are trademarks or registered trademarks of Research in Motion Limited.

Google App Engine, Google Apps, Google Checkout, Google Data API, Google Maps, Google Mobile Ads, Google Mobile Updater, Google Mobile, Google Store, Google Sync, Google Updater, Google Voice, Google Mail, Gmail, YouTube, Dalvik and Android are trademarks or registered trademarks of Google Inc.

INTERMEC is a registered trademark of Intermec Technologies Corporation.

Wi-Fi is a registered trademark of Wi-Fi Alliance.

Bluetooth is a registered trademark of Bluetooth SIG Inc.

Motorola is a registered trademark of Motorola Trademark Holdings LLC.

Computop is a registered trademark of Computop Wirtschaftsinformatik GmbH.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP BusinessObjects Explorer, StreamWork, SAP HANA, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects Software Ltd. Business Objects is an SAP company.






Sybase and Adaptive Server, iAnywhere, Sybase 365, SQL Anywhere, and other Sybase products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Sybase Inc. Sybase is an SAP company.

Crossgate, m@gic EDDY, B2B 360°, and B2B 360° Services are registered trademarks of Crossgate AG in Germany and other countries. Crossgate is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help → General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
<i>Example text</i>	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
Example text	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE TEXT	Keys on the keyboard, for example, F2 or ENTER.

Table of Contents

SAP NetWeaver Gateway plug-in for Eclipse	7
Documentation Types	8
Target Audience.....	8
Installing SAP NetWeaver Gateway plug-in for Eclipse.....	9
Post Installation Configuration	11
Enabling HTTPS for Secure Communications	13
Uninstalling Framework and Toolkits	16
Using the Tools of the Framework.....	17
Browsing SAP NetWeaver Gateway Services.....	18
Working with the Toolkits.....	24
Using the Java Platform Standard Edition Toolkit	25
Generating a JAVA SE Starter Application	26
Generated Basic Application	28
Generating Proxy for Java SE	31
Using the PHP Toolkit.....	37
Generating a PHP Starter Application.....	39
Generating a Proxy for a PHP Application	46
Creating Your Custom PHP Template	52
Using the Android Toolkit.....	54
Generating a Proxy for an Android Application	64
Working with the Sybase Unwired Platform Server	74
Creating Your Own Custom Android Template.....	84
Using the HTML5 Toolkit	86
Installing the HTML5 Toolkit.....	86
Generating an HTML5 Starter Application	87
Extending the Generated HTML5 Starter Application	91
Running your HTML5 Application	95
Security in your HTML5 Application	98
Extensibility.....	99
Overview of Extensibility	100
Implementing a New Template	101
Extending the New Template	104
Modifying a Template of a Toolkit	106
Example: Editing Your Custom Template	117
Implementing a New Environment.....	119
Extending the New Environment.....	121
Troubleshooting.....	123
General	123

Troubleshooting Java Platform Standard Edition Toolkit.....	123
PHP Toolkit Troubleshooting	125
PHP Runtime Issues	126
Troubleshooting the Android Toolkit	128
Android Runtime Issues	128
Troubleshooting the HTML5 Toolkit.....	129
Terminology	130

SAP NetWeaver Gateway plug-in for Eclipse

The SAP® NetWeaver® Gateway plug-in for Eclipse comes with a set of tools that enables you to generate code for applications for different target environments.

The application you create in the framework plug-in can retrieve, and interact with data from your existing SAP systems through SAP NetWeaver Gateway.

For more information about SAP NetWeaver Gateway, go to *help.sap.com* → *SAP NetWeaver Gateway*.

The framework plug-in enables developers working in Eclipse to do the following:

- Create and use templates for generating starter applications for supported target extensions.
The generated application interfaces with OData compliant SAP services, including generating code for Android, Java, PHP, and proxies.
- Generate proxies for OData compliant SAP services in a supported target environment.
- View, explore and discover OData compliant SAP services in a specific SAP NetWeaver Gateway system.

Supported Toolkits

The SAP NetWeaver Gateway plug-in for Eclipse provides a set of toolkits and allows you to create and use your custom toolkit.

Each toolkit defines a specific target environment and provides the tools for generating code for the runtime of applications in that environment.

When installed, the tools in the supported toolkit, such as, the target environment and templates, are available as part of the wizard that helps you to generate code for your SAP solution.

Currently, the supported toolkits in SAP NetWeaver Gateway plug-in for Eclipse include the following:

- SAP NetWeaver Gateway plug-in for Eclipse – Java Platform, Standard Edition Toolkit.
- SAP NetWeaver Gateway plug-in for Eclipse – PHP Toolkit.
- SAP NetWeaver Gateway plug-in for Eclipse – Android Toolkit.
- SAP NetWeaver Gateway plug-in for Eclipse – HTML5 Toolkit.

Exploring the OData Compliant SAP NetWeaver Gateway Services

Integrated into the framework, is the Search Console, which has been extended to provide you with the means to quickly search for specific Gateway services or explore and discover the various SAP services exposed in a specific SAP NetWeaver Gateway system.

For information on viewing, exploring and discovering SAP services, refer to: wiki.eclipse.org/E4/Search_Console

For more information, see *Starting the Search Console*.

SAP NetWeaver Gateway Java Library

The SAP NetWeaver Gateway Java library is a set of application programming interfaces (APIs) with which you can add simple or sophisticated OData compliant SAP service interfaces to your application source or proxies generated in the framework.

For information on using the SAP NetWeaver Gateway Java library, see the guide, *SAP NetWeaver Gateway Java Library*.

Documentation Types

The following is a list of the types of documentation available for SAP NetWeaver Gateway plug-in for Eclipse:

Document Type	Explanation
SAP NetWeaver Gateway plug-in for Eclipse	<p>This guide provides information on how to use the framework to generate code for client applications in various extensions to interact with data from your existing SAP systems, through SAP NetWeaver Gateway.</p> <p>Target group: Application developers, project teams, and system administrators.</p> <p>Current version: SAP NetWeaver Gateway 2.0 Support Package 3.</p>
SAP NetWeaver Gateway Java Library	<p>The SAP NetWeaver Gateway Java Library provides examples for using various classes and methods to call SAP services through SAP NetWeaver Gateway.</p> <p>Target group: Application developers, solution consultants, and project teams for implementations.</p> <p>Current version: SAP NetWeaver Gateway 2.0 Support Package 3.</p>
SAP NetWeaver Gateway plug-in for Eclipse Help	<p>Installed with the framework plug-in, is the help information which is similar to this guide.</p>

Target Audience

The information in this guide is relevant for:

- Developers working in Eclipse IDE.
- System administrators.
- Consultants, Solution Architects and Managers

Installing SAP NetWeaver Gateway plug-in for Eclipse

You can obtain the installation package for SAP NetWeaver Gateway plug-in for Eclipse to deploy in Eclipse from the SAP Community Network (SCN) site at:

<http://www.sdn.sap.com/irj/scn/downloads?rid=/webcontent/uuid/b09d414f-f227-2f10-bdbf-ba31c844b432>


You must install the framework together with at least one of the supplied toolkits, as the toolkit provides the environment and templates for your use.

Later, you can install other toolkits separately when you want.

Software Prerequisites

Ensure that the following prerequisites are met. You have:


- Installed Eclipse® Classic 3.6.2 or higher (Helios 3.6.2 and higher, or Indigo 3.7.1 and higher)
- Operating System:
Windows 7, Linux kernel 3.0 (Ubuntu 11.1), Mac – Snow Leopard 10.6.8
- SAP NetWeaver Gateway 2.0 SP0 and higher
 - The connection settings to an SAP NetWeaver Gateway server, including, the host name, and the port number.
 - User credentials for logging into the SAP NetWeaver Gateway server to which you want to connect.

 Note: The software prerequisite information of each toolkit is available in the section for using the specific toolkit in this guide.

Installation

First, download the installation package to your local file system, and install the framework and the toolkits as follows:

1. Start Eclipse, and from the main menu choose **Help → Install new software**. The Install dialog displays.
2. Click **Add**, and then choose **Archive**.
3. Select the zip file, **SAPNetWeaverGatewayPluginForEclipse.zip**, from the file system location where it was downloaded.
4. Choose **OK**, and select the option to install SAP NetWeaver Gateway plug-in for Eclipse–Framework (Required), and one or more of the following toolkits:
 - SAP NetWeaver Gateway plug-in for Eclipse – Java Platform, Standard Edition Toolkit.
 - SAP NetWeaver Gateway plug-in for Eclipse – Android Toolkit.
 - SAP NetWeaver Gateway plug-in for Eclipse – PHP Toolkit.
 - SAP NetWeaver Gateway plug-in for Eclipse – HTML5 Toolkit.
5. Accept the license to continue with the installation.
6. On completion, Eclipse prompts you to restart. Restart Eclipse.

 Note: The installation information for each toolkit is available in the section for using the specific toolkit in this guide.

After you have restarted Eclipse, you can check the installation.

To verify that your installation displays in the software installed list, go to the main menu and select **Help** → **About Eclipse SDK** → **Installation Details** → **Plug-ins**.

Post Installation Configuration

You must enable the wizards to provide you with a list of the SAP NetWeaver Gateway hosts from which you can choose the SAP service you want.

When configuring the SAP NetWeaver Gateway connection, you can specify which one is the default connection.

If there is only one connection, it is set as default automatically.

Only one connection can be set as default, and a default connection must always be specified. The word "(default)" is appended to the host name.

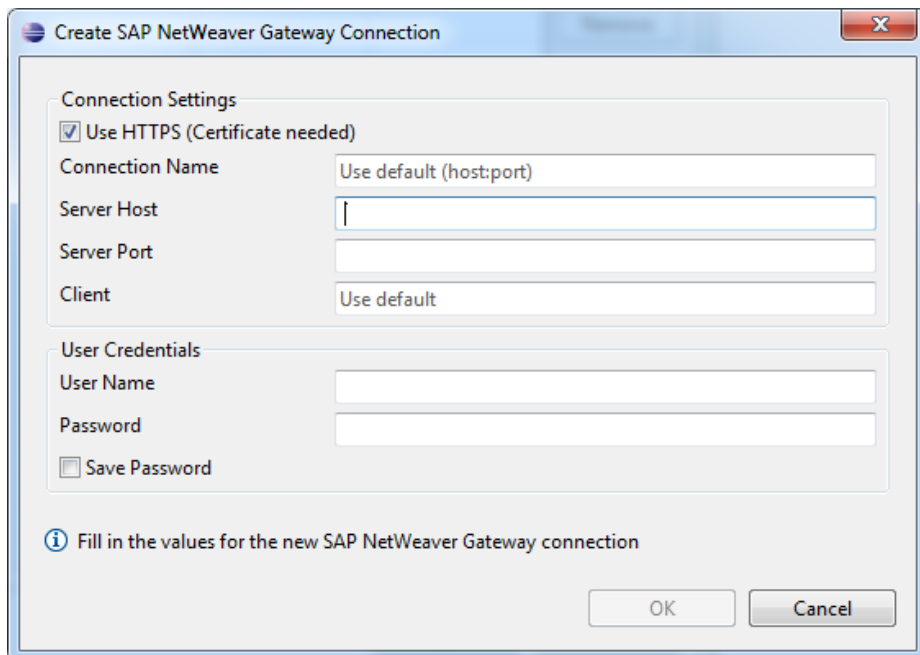
 **Note:** You can perform the post installation configuration when running a wizard.

To configure the connection settings for an SAP NetWeaver Gateway host in Eclipse:

1. Choose **Window** → **Preferences** from the Eclipse main menu.
2. Select **SAP NetWeaver Gateway** → **Connections**. The SAP NetWeaver Gateway Destinations dialog displays.

If you have already provided connection settings to an SAP NetWeaver Gateway host, those details are presented.


If you are using the wizard for the first time, choose **Add**. The Create SAP NetWeaver Gateway Connection displays.



3. Enter the following connection settings for an SAP NetWeaver Gateway host:
 - **Use HTTPS (Certificate needed):**
This option is selected by default, to enable you to use secure connection with authentication in the host.
 - **Requires that Secure Socket Layer (SSL) has been enabled in your SAP NetWeaver Gateway landscape.**
 - **Connection Name:**
Specify a name for the SAP NetWeaver Gateway host. This is optional. If you do not provide a name, the host name and the port number will be used.
 - **Server Host:**
Specify the host name or IP address for the SAP NetWeaver Gateway host.

- **Server Port:**
Specify the port number for the SAP NetWeaver Gateway host.
- **Client:**
Specify the client number of the SAP NetWeaver Gateway host. This is optional. If you do not specify a client, the default client number of the specified host name is used.
- **User Credentials:**
Logon information of an SAP NetWeaver Gateway user.
- **User Name:**
Specify the user name for logging onto the SAP NetWeaver Gateway host.
- **Password:**
Specify the password for the user.
- **Save Password:**
Saves the logon credentials for subsequent logon to the specified host when you run the framework.

4. Choose **OK**.


 **Note:** The relating post installation configuration for each toolkit is available in the section for using the specific toolkit in this guide.

Enabling HTTPS for Secure Communications

You can enable the use of the HTTPS protocol for secure communications where the information that is exchanged between your application and SAP NetWeaver Gateway server is sensitive.

First, configure the use of SSL in the SAP NetWeaver Gateway landscape, and then configure SSL in framework.

The following is an overview of how to enable HTTPS protocol for secure communications:

1. Obtain and install in your work station, the root certificate (CA root certificate) of the SAP NetWeaver Gateway server that has been configured as the SSL server.
 2. Add information about the SSL server certificate to the Java Runtime Environment (JRE) Keystore using the application, Keytool.
-  **Note:** You can obtain the CA root certificate directly from the system administrator of the SAP NetWeaver Gateway server.

Obtain and Install the CA certificate from the SAP NetWeaver Gateway server

To export Gateway's SSL server certificate from SSL Sever Standard PSE:


1. Use the Trust manager (transaction **STRUST**) in the SAP NetWeaver Gateway system to export the CA's Root certificate.
2. Select **SSL Server Standard** node in right-hand side tree
3. Choose the **SSL certificate** under **Own Certificate**, choose **Certificate**, and then choose **Export**.
4. Specify the location of the certificate in your file system.

Adding Information about the Gateway Server Certificate to Your Keystore

Download the utility, **Keytool** to help you create and manage digital certificates. You can obtain the utility at: <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html>

Keytool is a command-line utility that allows you to create and manage keystores for digital certificates in the Java environment.

You can list the current certificates contained within the keystore using the -list command of the keytool. The initial password for the `cacerts` keystore is *changeit*.

 For example:

```
C:\Program Files\Java\jdk1.6.0_26\jre\bin>keytool -list -keystore  
..\lib\security\cacerts
```

Enter keystore password: *changeit*. The following displays:

```
Keystore type: jks  
Keystore provider: SUN  
Your keystore contains 11 entries:  
engweb, Wed Apr 11 16:22:49 EDT 2001, trustedCertEntry,  
Certificate fingerprint (MD5):  
8C:24:DA:52:7A:4A:16:4B:8E:FB:67:44:C9:D2:E4:16  
thawtepersonalfreemailca, Fri Feb 12 15:12:16 EST 1999, trustedCertEntry,  
Certificate fingerprint  
(MD5):1E:74:C3:86:3C:0C:35:C5:3E:C2:7F:EF:3C:AA:3C:D9  
thawtepersonalbasicca, Fri Feb 12 15:11:01 EST 1999, trustedCertEntry,
```

```

Certificate fingerprint (MD5):
E6:0B:D2:C9:CA:2D:88:DB:1A:71:0E:4B:78:EB:02:41
verisignclass3ca, Mon Jun 29 13:05:51 EDT 1998, trustedCertEntry,

Certificate fingerprint (MD5):
78:2A:02:DF:DB:2E:14:D5:A7:5F:0A:DF:B6:8E:9C:5D
thawteserverca, Fri Feb 12 15:14:33 EST 1999, trustedCertEntry,

Certificate fingerprint (MD5):
C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
thawtepersonalpremiumca, Fri Feb 12 15:13:21 EST 1999, trustedCertEntry,

Certificate fingerprint (MD5):
3A:B2:DE:22:9A:20:93:49:F9:ED:C8:D2:8A:E7:68:0D
verisignclass4ca, Mon Jun 29 13:06:57 EDT 1998, trustedCertEntry,

Certificate fingerprint (MD5):
1B:D1:AD:17:8B:7F:22:13:24:F5:26:E2:5D:4E:B9:10
verisignclass1ca, Mon Jun 29 13:06:17 EDT 1998, trustedCertEntry,

Certificate fingerprint (MD5):
51:86:E8:1F:BC:B1:C3:71:B5:18:10:DB:5F:DC:F6:20
verisignserverca, Mon Jun 29 13:07:34 EDT 1998, trustedCertEntry,

Certificate fingerprint (MD5):
74:7B:82:03:43:F0:00:9E:6B:B3:EC:47:BF:85:A5:93
thawtepremiumserverca, Fri Feb 12 15:15:26 EST 1999, trustedCertEntry,

Certificate fingerprint (MD5):
06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
verisignclass2ca, Mon Jun 29 13:06:39 EDT 1998, trustedCertEntry,


Certificate fingerprint (MD5):
EC:40:7D:2B:76:52:67:05:2C:EA:F2:3A:4F:65:F0:D8

```

Adding the CA Root Certificate to the Keystore

You must add the CA certificate you received from the SAP NetWeaver Gateway server to the Eclipse keystore.

From the command line, enter `keytool -import`, to import the file into your `cacerts` keystore.

 For example:

```

C:\Program Files\Java\jdk1.6.0_26\jre\bin>keytool -import -keystore
..\lib\security\cacerts -file c:\ SAProotca.cer

```

To check, run **keytool -list** again to verify that your private root certificate was added. For example,

```

C:\Program Files\Java\jdk1.6.0_26\jre\bin>keytool -list -keystore
..\lib\security\cacerts

```

You should now see a list of all the certificates including the one you just added. In addition, verify that the java home location is defined in the file, *eclipse.ini*. For example,

```

-vm
C:/Program Files/Java/jdk1.6.0_21/bin/javaw.exe

```

Configuring HTTPS in the Framework

After you have configured the use of SSL in the SAP NetWeaver Gateway landscape, you can configure SSL in the framework.

To configure SSL in the framework:

1. From the main menu, select **Window** → **Preferences** → **SAP NetWeaver Gateway** → **Connections**. The SAP NetWeaver Gateway Destinations dialog displays.
2. Choose **Add**. The Create Gateway Destinations displays.
3. Select **Use HTTPS (Certificate needed)**, and enter the connection settings for the SAP NetWeaver Gateway host.

Uninstalling Framework and Toolkits

You can remove the SAP NetWeaver Gateway Plug-in for Eclipse - Framework and the supported toolkits.

To remove SAP NetWeaver Gateway Plug-in for Eclipse - Framework and the supported toolkits:

1. Start Eclipse, and from the main menu, and choose **Help → About Eclipse SDK**. The About Eclipse SDK window opens.
2. Choose **Installation Details → Installed Software**. The list of installed software components display.
3. Select the plug-in you want to remove and choose **Uninstall**.

Using the Tools of the Framework

SAP NetWeaver Gateway plug-in for Eclipse - Framework provides wizards that obtain user actions and data in order to generate code suitable for the specific supported extension.

The framework enables you to:

- Browse, view, and identify the SAP services you can use for various business solutions in a specific SAP NetWeaver Gateway system.

For more information, see *Browsing SAP NetWeaver Gateway Services*, on page 18.

- Use and create your custom toolkit to generate starter applications or proxies that interface with SAP services in the target extensions.
- Generate source code for a proxy that calls an SAP service to interface with your existing applications in a supported target environment through an SAP NetWeaver Gateway system.

Browsing SAP NetWeaver Gateway Services

The SAP NetWeaver Gateway plug-In for Eclipse - Framework is integrated with a search console that allows you to explore and discover OData compliant SAP service interfaces available in an SAP NetWeaver Gateway system.

Use the search provider to search for SAP service in a specific SAP NetWeaver Gateway landscape.

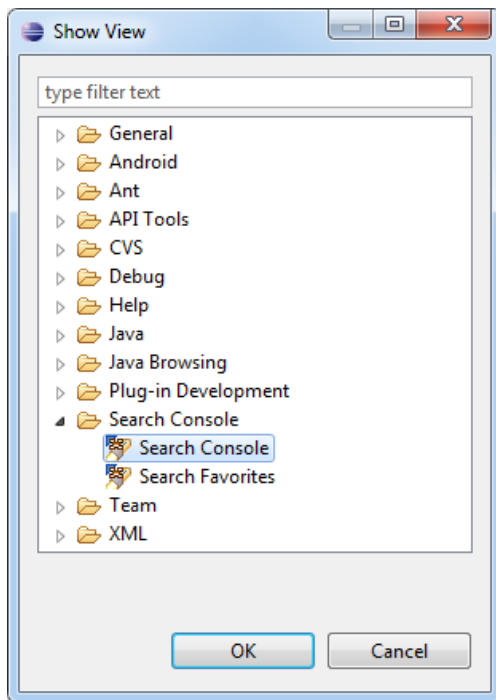
You define your search criteria and browse through the results that are displayed. Later, you can call the SAP service interfaces you have discovered in the applications generated for supported target environment.

Starting the Search Console

You start the search console to search for SAP service interfaces in an SAP NetWeaver Gateway landscape.

To start the Search Console:

1. From the Eclipse menu, select **Window** → **Show View** → **Others**.
4. Browse to select **Search Console**.



The Search Console tab displays in your Eclipse workspace.

5. Click **OK**.

You can define your search criteria and browse through the results that are displayed.

Specifying Gateway Systems

You can add to the Destinations list, connection settings of the SAP NetWeaver Gateway system in which you want to perform your search.

For more information, see *Post Installation Configuration*.

Finding SAP Service Interfaces

You define the search item in the search console and press `Enter` or click the *Search* button to search in the specified SAP NetWeaver Gateway environment for the service interfaces matching your query.

Your search item or query can be the exact name of the service interface you are looking for, or part of a name, or simply use the wildcard character, `*` to get all services.

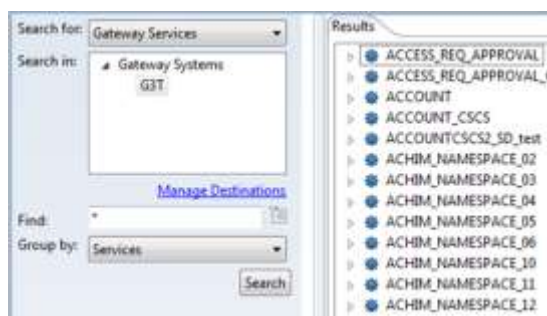
If you search for `user`, you will get all services with their names containing the substring `user`.

The wildcard character can be used for searching for all services, or for searches such as: *workflow*processing*. The results will include items such as *workflowprocessing*, *workflow_processing*, or *workflowdecisiontaskprocessing*.

The items in the query are case insensitive, and must not contain punctuations, including, `@`, `#`, `$`, `%`, `^`, `&`, `(`, `)`, `=`, `+`, `[`, `]`, `\`, `?`, and other special characters.

To perform a search for an SAP service:

1. Start the Search Console.
2. In Search for, select **Gateway Service Search**. All the Gateway systems that have been configured to connect to the framework are listed.



3. In Search in, select the **Gateway system** in which to perform the search. For example, G3T.
4. In Find, enter the query or search item, and in Group by, choose **Services**.
5. Click **Search** to start searching. Retrieving search results may take some time.

Viewing your Search Results

You can browse the search results, select, and navigate the results pane.

The search results are provided in the same page as the search item. That way, you can see the search item and the results at the same time. If you are not satisfied with the results, you can edit the search item to perform a new search.

The results are presented as a tree of services, and each represents a service interface, and when you expand it, you can explore its collection and entity sets.

Selecting a service displays the collections and the function imports in the service.

In addition, the Details tab displays. You can find detailed information about the service, including, description, technical and service names, Identifier, title, date and others.

The screenshot shows a software interface with a search bar on the left and a results pane on the right. The search bar contains 'Gateway Services' and 'GRT'. The results pane shows a list of services, with 'WFSERVICE' selected. The 'Details' tab is active, displaying the following information:

Details	
Description	Workflow service for SP3 or higher
UpdatedDate	2011-12-05T15:19:50
ID	2WFSERVICE_0001
Author	TEHRE
ServiceUrl	http://www.BEIS.wdf.sap.corp:50000/sap/opu/odata/wwk/WFSERVICE
ImageUrl	
TechnicalServiceName	/WWW/WFSERVICE
Title	WFSERVICE
MetadataUrl	http://www.BEIS.wdf.sap.corp:50000/sap/opu/odata/wwk/WFSERVICE/\$metadata
TechnicalServiceVersion	1

The results pane also displays a list of collections and function imports for the selected service:

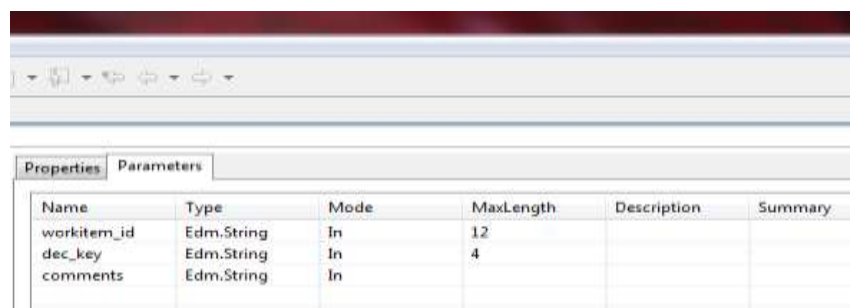
- AttachmentCollection
- BusinessObjectAttachmentCollection
- CommentCollection
- DecisionOptionCollection
- ExtensibleElementCollection
- ExtensibleElementUpdateCollection
- InitiatorCollection
- NotificationCollection
- OutboxTaskCollection
- ParticipantCollection
- PossibleAgentCollection
- RelatedObjectCollection
- RequesterDataCollection
- StepHistoryOutputCollection
- SubscriptionCollection
- SubstitutionCollection
- SubstitutionProfileCollection
- TaskDescriptionCollection
- WorkflowTaskCollection
- ActivateSubstitution
- ApplyDecision
- ApplyForward
- ApplyRelease
- ApplyReverse
- DeactivateSubstitution
- GetStepHistory
- GetSubstitutesByUserContext

When you select a function import, it displays the following tabs:

- Properties: Specifies the name, the return type, and the method.



- Parameters: Specifies the following for the entity type of the function import:
- Name: Specifies the name of the property.
- Type: Specifies the type of property
- Mode: Specifies whether it is an input or output parameter.
- MaxLength: Specifies the number of characters allowed for the entity type value.
- Description: Specifies the descriptive text of the entity type.
- Summary: Specifies short information about the entity type.



Selecting a collection within a service displays the following:

- Properties tab:

The Properties tab specifies the URL of the selected collection. When you click the URL, it opens the default Web browser to present the metadata of the collection.

On the right hand, is a pane that displays the annotations included in the service.

In addition, it displays the entity type properties grouped into an entity set. The following are the attributes of each property:

- Name: Specifies the name of the property.
- Label: Specifies the label of the property.
- Type: Specifies the type of property.

Properties			Associations	Entity Description
http://sap/opu/odata/iwwrk/WFSERVICE/WorkflowTaskCollection				
Name	Label	Type		
workitem_id	ID	Edm.String		
status	Status Code	Edm.String		
status_txt	Status Description	Edm.String		
subject	Subject	Edm.String		
type	Type	Edm.String		
priority	Priority	Edm.String		
task_name	Task Bound Item type	Edm.String		
actual_owner	Actual Owner	Edm.String		
actual_owner_name	Actual Owner Name	Edm.String		
note_count	No. of Attachments	Edm.Int32		
reassign_by	Forwarder	Edm.String		
reserved_by	Reserved By	Edm.String		
act_dec	Decision Key	Edm.String		
act_dec_agent	Decision Agent	Edm.String		
language	Language	Edm.String		
start_dl	Start Deadline	Edm.DateTime		
end_dl	End Deadline	Edm.DateTime		
created_at	Created At	Edm.DateTime		
created_by	Creator	Edm.String		
gui_link	WebGUI link	Edm.String		
mime_type	MIME Type	Edm.String		

Annotations:
MaxLength: 120
sap:sortable: false
sap:updatable: false
sap:filterable: false

- **Associations:**
Describes the entity type navigation properties. The following attributes describe each navigation property:
 - **Name:** Specifies the name of the navigation property.
 - **From:** Specifies the entity type to navigate from, using this navigation property. In addition, it is the entity type of the selected entity set.
 - **Multiplicity:** Specifies the cardinality of the **From** part of the association.
 - **To:** Specifies the entity type to navigate to, using this navigation property.
 - **Multiplicity:** Specifies the cardinality of the **to** part of the association.
- **Entity Description:**
Specifies the attributes and details of the entity set.
- **Key:** Specifies the attribute name of the entity set. Each attribute of the selected entity set is described using key and value.

Value: Specifies the corresponding value of the entity set attribute. <no value> if it is not specified for the entity set.

Results		
<ul style="list-style-type: none"> WFDECISIONTASKPROCESSING WFHUBSERVICE WFMGWPROCESSING WFODCPROCESSING WFSERVICE <ul style="list-style-type: none"> AttachmentCollection BusinessObjectAttachmentCollection CommentCollection DecisionOptionCollection ExtensibleElementCollection ExtensibleElementUpdateCollection InitiatorCollection NotificationCollection OutboxTaskCollection ParticipantCollection PossibleAgentCollection RelatedObjectCollection RequesterNoteCollection StepHistoryOutputCollection SubscriptionCollection SubstitutionCollection SubstitutionProfilesCollection TaskDescriptionCollection WorkflowTaskCollection ActivateSubstitution ApplyDecision ApplyForward ApplyRelease ApplyReserve DeactivateSubstitution GetStepHistory GetSubstitutesByUserContext 	Properties	Associations
	Entity Description	
	Key	Value
	Entity Type SA...	1
	SAP Deletable	false
	SAP Pageable	false
	SAP Updatable	false
	SAP Creatable	false
	EntitySet Name	WorkflowTask...
	Entity Set SAP ...	1
	SAP Searchable	true
	SAP Queryable	false
	URL	http://vmw38...
	Entity Type	WorkflowTask

Generating Proxy for a Service

You can quickly generate a proxy for the SAP service you select.

To do so:

1. Start the search console, and locate the service you want.
2. Right click the service, and choose **Generate Proxy**.
If the service is not well formed, you cannot perform this step.
The Proxy Generation wizard opens.
3. Click **Browse** to select the project in which to generate the proxy.
4. From **Create proxy for**, select the target extension or toolkit for which you want to create the proxy, and specify the package name.
5. Click **Finish**, to generate the proxy. If you choose **Next**, the SAP NetWeaver Gateway host and the URL of the selected service are provided.

Working with the Toolkits

The SAP NetWeaver Gateway plug-in for Eclipse Framework provides you with a comprehensive set of toolkits for developing and delivering SAP solutions and applications in your specified target extension platform. The following are the supplied toolkits:

- Java Platform Standard Edition (Java SE) Toolkit
- PHP Toolkit
- Android Toolkit
- HTML5 Toolkit

Each toolkit is a collection that consists of an environment, a pattern and templates to enable you to rapidly develop and bring SAP solutions and applications to market.

In addition, the framework is extensible, allowing you to create the following:

- Environments
- Templates

Packaging your custom environment and its associated templates together, you create a custom toolkit through the extensible processing system of the framework.

When you make your custom toolkit available to other developers, they can use it to develop and deliver SAP solutions and applications in the specific target environment.

For more information, see *Extensibility*.

Using the Java Platform Standard Edition Toolkit

The SAP NetWeaver Gateway plug-in for Eclipse – Java Platform, Standard Edition Toolkit is for developing and delivering SAP solutions and client applications for the Java platform.

Software Prerequisites

Ensure that the following prerequisites are met. You have:

- SAP NetWeaver Gateway 2.0 or higher
 - The connection settings to an SAP NetWeaver Gateway server. For example, the host name, and the port number.
 - User credentials for logging into the SAP NetWeaver Gateway server to which you want to connect.
- Downloaded and installed the SAP NetWeaver Gateway plug-in for Eclipse Framework.

Installing the Java SE Toolkit


Browse the installation package in your local file system to locate the target environment as follows:

1. Start Eclipse, and from the main menu choose **Help → Install new software**. The Install dialog displays.
2. Click **Add**, and then choose Archive.
3. Select the zip file, **SAPNetWeaverGatewayPluginForEclipse.zip**, from the file system location where it was downloaded.
4. Choose **OK**, and select SAP NetWeaver Gateway plug-in for Eclipse – Java Platform, Standard Edition Toolkit.
Accept the license to continue with the installation.
5. On completion, Eclipse prompts you to restart. Restart Eclipse.

Generating a JAVA SE Starter Application

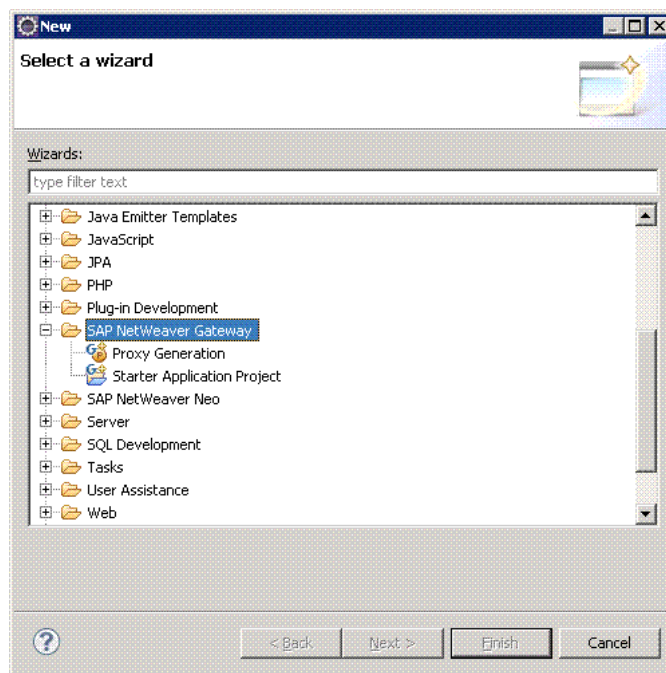
You can create a Java SE starter application that calls SAP services in SAP NetWeaver Gateway.

Using the project wizard, you generate code into the new Java SE project that you specify.

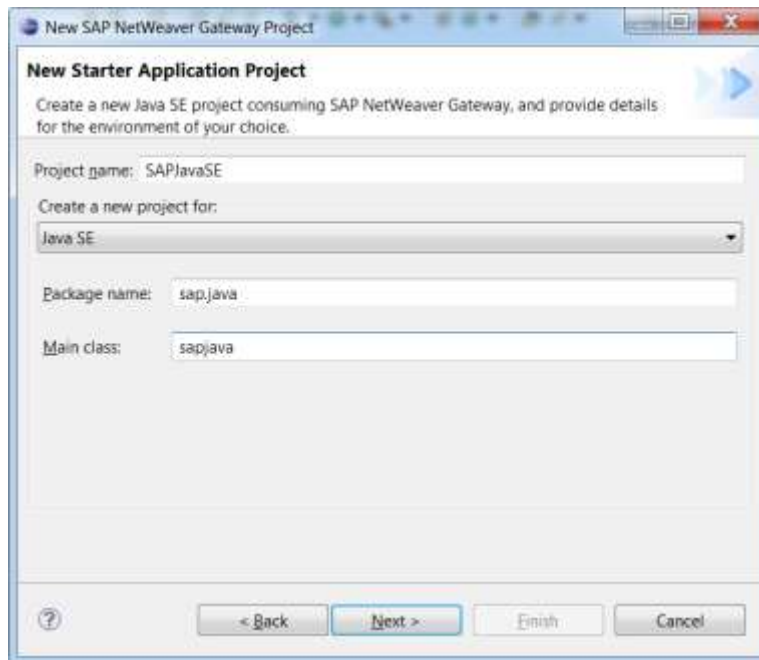
 **Note:** Using the Java SE toolkit, you can only generate code for a custom application.

To create your application:

1. From the File menu, choose **New** → **Other** → **SAP NetWeaver Gateway**, or from the keyboard press **CTRL+N**, and select **Starter Application Project**. The New wizard dialog displays.



6. Enter a new name for your project. The project and the relevant resources will be created for only your application. For example, SAPCRMCONTACTJSE.
7. From **Create new project for**, select **Java SE**.
A pane for specifying the details of the application as required by the selected toolkit opens.
8. Specify a name for the Package and the Main class.



9. Choose **Next**. The page for the templates for the supported toolkit displays.
Custom Application template is the only template available for your Java SE application.
10. Click **Finish** to generate the code.

Generated Basic Application

A Java class file is created using the package name you have defined.

Use the basic application template to familiarize yourself with writing code that calls the SAP NetWeaver Gateway Java library provided in the framework, to consume SAP NetWeaver Gateway services.

The generated project consists of the following:

- SRC folder
This folder contains the specified package with a Java file that contains the main() method for calling and initiating the Java library
- JRE system library
Contains the Java Platform, Standard Edition library components.
- LIB folder
Contains the SAP NetWeaver Gateway Java library component.

Extending the Generated Basic Application

Use the generated basic application to develop your own application using the Java library API to call SAP NetWeaver Gateway.

The following is an example of how to use the generated basic application:

First, determine the business scenario for your application. For example, your application will do the following:

“Enable users to search for flights from one location to another”

Users should be able to get details about a flight, such as, city of origin, destination city, departure date, and arrival date.

- ❏ The following is an example code that calls the Java library to consume the SAP service, *RMTSAMPLEFLIGHT* in an SAP NetWeaver Gateway server:

- Creating the request URL.

```
String requestUrl = "/sap/opu/sdata/iwfnd/RMTSAMPLEFLIGHT/CarrierCollection  
(carrid='" + carrid + "')";
```

- Defining Gateway connection settings.

```
IServerConnectionParameters gateway =  
new ServerConnectionParameters ("<host>", "<port>", "<client id>", <useSSL>);
```

- Create the authentication object.
The example below shows basic authentication implementation.

```
IAuthentication credentials = new  
UsernamePasswordCredentials("<user>", "<password>");
```

Note: You must replace *<user>*, *<password>* with the appropriate values.

You can add your own interceptors and authentication implementation.

Initializing the REST Client based on the connection details. The following are the option for setting the representation: ATOM, or JSON.

```
IRestClient restClient = RestClientFactory.createInstance(gateway,  
credentials, Representation.ATOM);
```

- Executing the request.



```
IRestRequest request = new RestRequest(requestUrl);
IRestResponse response = restClient.execute(request);
return response.getBody();
```

Example code for parsing returned response

- Parsing the response string XML into an ODataEntry object.



```
ODataEntry carrierEntry =
ODataParserFactory.createInstance(Representation.ATOM).parseODataEntry(response);
```

- Getting the carrier properties and printing to the standard output.



```
ODataProperty[] properties = entry.getProperties();
for (ODataProperty oDataProperty : properties)
{
    System.out.println(oDataProperty.getName() + ": " +
        oDataProperty.getValue());
}
```

Example code for using a function import: GetAvailableFlights.

- Create the request URL.



```
String requestUrl =
"/sap/opu/sdata/iwfnd/RMTSAMPLEFLIGHT/GetAvailableFlights?cityfrom="
+ cityFrom
+ "&cityto="
+ cityTo
+ "&fromdate="
+ fromDate
+ "&todate=" + toDate;
```

- Defining Gateway connection details.



```
IServerConnectionParameters gateway =  
new ServerConnectionParameters("<host>", "<port>", "<client_id>",  
                                "<useSSL>");
```

- Initializing the REST client based on the connection settings.



```
IRestClient restClient = RestClientFactory.createInstance(gateway,  
credentials, Representation.ATOM);  
IAuthentication credentials = new  
UsernamePasswordCredentials("<user>", "<password>");
```

- Calling the SAP NetWeaver Gateway server.

```
IRestRequest request = new RestRequest(requestUrl);  
IRestResponse response = restClient.execute(request);  
return response.getBody();  
}
```

Generating Proxy for Java SE

You can create a proxy for a supported toolkit to call SAP services through your SAP NetWeaver Gateway server.

The proxy provides the implementation interfaces for the SAP service you choose to use.

You can create your own application to interface with the generated proxy, or you can create a starter application based on the Basic Application template to interface with the generated proxy.

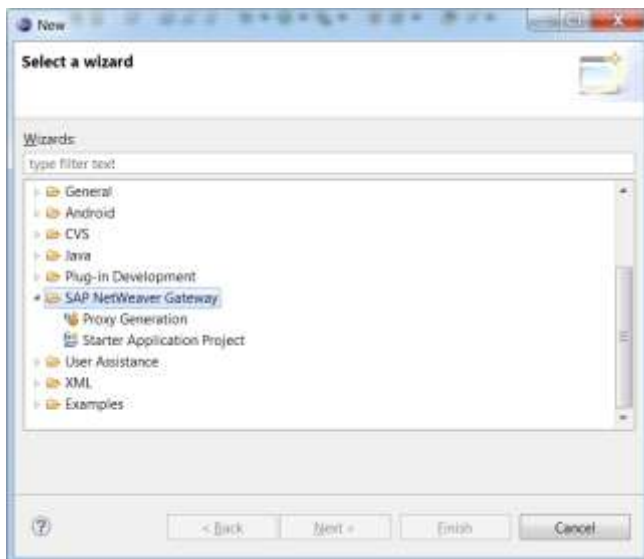
Requirements

Make sure that you have a project for the toolkit (target environment) in which you want to generate the proxy.

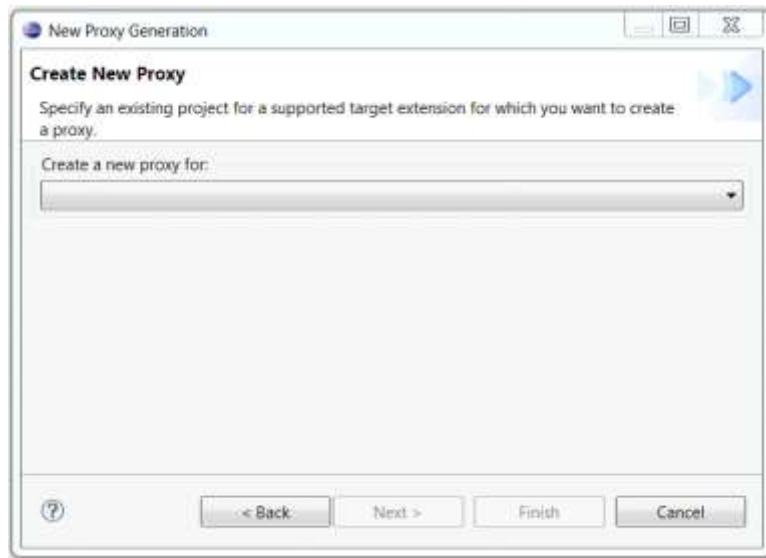
You get a warning message when you attempt to generate code for an environment that is different from the target environment of the selected project.

To create your proxy:

1. From the File menu, choose **New** → **Other** → **SAP NetWeaver Gateway**, or from the keyboard press **CTRL+N**, and then select **Proxy Generation**. The New wizard dialog displays.



2. From **Create new proxy for**, selected **Java SE**.
3. Select an existing project for the toolkit for which you want to create the proxy.
Depending the properties of the environment for the selected toolkit, you can generate the code into a project or a folder in the file system.
4. Specify the Package name.
You can use the default package option without specifying a package name



5. Choose **Next**. The page for specifying the SAP service you want your application to call displays.




6. Select **Remote location**, to connect to your SAP NetWeaver Gateway server within the network.
7. Click **Catalog** to search for the specific SAP service. The exploration dialog opens for you to search for the service.

When you click **Catalog**, without providing a specific URL, the specified default SAP NetWeaver Gateway is automatically selected.

If you have not saved the password of the default SAP NetWeaver Gateway, a popup displays requesting the password.

Alternatively, select **File system**, and click **Browse** to specify the paths to both the service metadata document for the specific service, and the service document in your file system.

Click **Validate** to verify that the service documents are properly formed XML files for OData compliant services.

 **Note:** If the specified URL of the service metadata comes from the Search Console, the Validate button is not available, as both the connection and the metadata document is verified.

The Validate button is available only if you have manually entered the URL, or selected the metadata file from the file system.

8. Choose **Finish**.

The proxy wizard generates code for the selected target environment into the specified Eclipse project.

In addition, it automatically creates class files for the selected SAP service . The number of files created depends on the number of entities in the selected SAP service.

Using the Generated Proxy for Java SE


Below is an example of the contents generated for the SAP service, *RMTSAMPLEFLIGHTService*.


The generated proxy consists of the following:

- SRC folder
This folder contains the proxy classes for the selected SAP service.
- JRE system library
Contains the Java Platform, Standard Edition library components.
- LIB folder
Contains the SAP NetWeaver Gateway Java library component.


You can call the generated proxy classes from your application to interface with the selected SAP service through the specified SAP NetWeaver Gateway server.

Note that, you have to create your own user interface for rendering your application.

 The following is an example of a class with a main method that calls the generated classes and methods for the SAP service, *RMTSAMPLEFLIGHTService*.

 Each line in the example has comments to help you understand the expected process:


- Initialize the service REST Client

```
  
IServerConnectionParameters serverConnectionDetails = new  
ServerConnectionParameters("<host>", "<port>", "<client>", <useSSL>);  
  
IAuthentication authentication = new  
UsernamePasswordCredentials("<username>", "<password>");  
  
IRestClient client =  
RestClientFactory.createInstance(serverConnectionDetails, authentication,  
Representation.ATOM);
```

- Create an instance of the service.

```
  
RMTSAMPLEFLIGHTService proxy = new RMTSAMPLEFLIGHTService(client);
```

- Filter flight collection according to airline.

```
  
List<Flight> flightCollection = service.getFlightCollection("$filter=carri  
eq 'AA'");  
System.out.println("Number of flights in the collection: " +  
flightCollection.size());
```

- Create a booking object and provide the details of the booking information for a specific flight of the carrier.



```
Date fldate = m_ISO8601Local.parse("2012-01-25T00:00:00");
Booking booking = new Booking("AA", "0017", fldate, "0", proxy );
booking.setCUSTOMID("00004617");
booking.setCOUNTER("00000000");
booking.setAGENCYNUM("00000325");
booking.setPASSNAME("Example Passenger Name1");
```

- Create a new booking entry in the booking collection for a specific flight.

It returns the newly created booking in Gateway with additional information, for example, the booking ID, and prints the booking ID of the newly created booking entry.



```
Booking booking2 = service.createBookingCollectionEntry(booking);
System.out.println("success: booking id: " + booking2.getbookid());
```

- Update an entry: You can change a booking entry; update the booking entry object, for example, by providing a new passenger name and calling to Gateway to update the booking entry.

If successful, it reads the booking entry from Gateway and prints the updated entry, passenger name.



```
booking2.setPASSNAME("Example Passenger Name2");
boolean updateSuccess = service.updateBookingCollectionEntry(booking2);
if (updateSuccess)
{
    Booking booking3 = service.getBookingCollectionEntry(booking2.getcarrid(),
    booking2.getconnid(), booking2.getfldate(), booking2.getbookid());
    System.out.println("booking3: " + booking3.getPASSNAME());
}
```

- Delete an entry: You can delete the booking entry you created and print the status.



```
boolean deletedSuccess =
service.deleteBookingCollectionEntry(booking3.getcarrid(),
booking3.getconnid(), booking3.getfldate(), booking3.getbookid());
if (deletedSuccess)
{
    System.out.println("Deleted");
}
}
```

- Read: Read the booking collection of the specific flight (navigation property) and print all the passenger names.



```
List<Booking> bookingCollection =
service.getFlightCollectionEntry(booking.getcarrid(), booking.getconnid(),
fldate).flightBookings();
for (Booking bookingEntry : bookingCollection) {
System.out.println("Passenger name - " + bookingEntry.getPASSNAME());
}
```

- Function import: Example for using the flight service specific method to get flights between specific dates and destination.



```
DateFormat formatter = new SimpleDateFormat("yyyyMMdd");
Date date = formatter.parse("20110420");
List<Flight> availableFlights = proxy.GetAvailableFlights(date, date, "NEW
YORK", "SAN FRANCISCO");
System.out.println(availableFlights.size());
} catch (ProxyException e) {
System.out.println(e.getMessage());
}
```

- Below is an example code for getting media links, based on the Gateway service, *Workflow*, which is different from the service used in the example above.

It obtains a list of workflow tasks from the Gateway server.



```
List<WorkflowTask> taskList = service.getWorkflowTaskCollection();
```

- Obtains a specific attachment.

This comes from the first task item in the list, and gets its attachment.



```
Attachment taskAttachment = taskList.get(1).Attachments().get(1);
```

- Gets the mime type of the attachment.



```
String taskAttachmentMimeType = taskAttachment.getMimeType();
```

- Obtains the byte buffer (a fixed-capacity buffer that holds byte values) of the media resource



```
byte[] bytes =
service.getMediaResourceBytes(taskAttachment.getMediaResource());
```

- Add your own code to use the media resource to suit your needs.

For example, if the mime type is a GIF, then write the byte buffer to a file with the GIF suffix.

Error and Exception handling in the Proxy

In your application, you should perform error handling and exceptions thrown by the proxy, (ProxyException), to suit your needs.

❏ For example, using the code above, you can display an error message such as:

```
❏  
} catch (ProxyException e) {  
System.out.println(e.getMessage());  
}
```

Displayed as: Booking AA 00002161 has already been canceled

Running your JAVA SE Application

You can run and test the generated Java SE application in Eclipse just as you run any Java application created in Eclipse.

Java SE Application Security

The framework automatically enables your application to make requests that include a challenge token and ensure that the requests from your application are not coming from another source other than the user.

Java SE Application Support for Cross Site Request Forgery Protection

SAP services are protected against Cross-Site Request Forgery (XCSRF) attacks. This protection mechanism appends challenge tokens to each request and associates them with each user's session. Each token is unique per user session, and per request.

❏ Note: All SAP services that are compatible with OData library version 1.0 are protected and require XCSRF tokens. Some services that are compatible with OData library 0.2 also require these tokens.

You can disable the protection using CSRF tokens by implementing your custom code. For more information, see the section, *Getting Started* in the SAP NetWeaver Gateway Java Library.

Using the PHP Toolkit

The SAP NetWeaver Gateway plug-in for Eclipse – PHP Toolkit is a collection of environment, pattern and a template, which are suitable for developing SAP solutions and applications for PHP.

In addition, you can use the extensible tools of the framework to create your custom templates.

For more information, see *Creating Your Custom PHP Template*

Software Prerequisites


Ensure that the following prerequisites are met. You have:

- Downloaded and installed the SAP NetWeaver Gateway plug-in for Eclipse Framework.

For SAP NetWeaver Gateway plug-in for Eclipse – PHP Toolkit; you require the following:

- PHP: Hypertext Preprocessor version 5.0 or higher.
- OData SDK for PHP; you can obtain the OData SDK for PHP at: odataphp.codeplex.com

The user guide is available in the installation folder of the OData SDK at:
..\doc\User_Guide.htm

 **Note:** If you do not install and configure OData SDK for PHP as mentioned in the user guide, Eclipse fails to generate the PHP application.

To run your generated PHP application, you need the following:

- PHP server.
- Operating System: Windows 7, Linux kernel 3.0 (Ubuntu 11.1), Mac – Snow Leopard (10.6.8)
- Browsers: Internet Explorer, Firefox, Chrome, or Safari

Installing the PHP Toolkit

Browse the installation package in your local file system to locate the target environment as follows:

1. Start Eclipse, and from the main menu choose **Help → Install new software**. The Install dialog displays.
2. Click **Add**, and then choose **Archive**.
3. Select the zip file, **SAPNetWeaverGatewayPluginForEclipse.zip**, from the file system location where it was downloaded.
4. Choose **OK**, and select the option to install SAP NetWeaver Gateway plug-in for Eclipse – PHP Toolkit.
5. Accept the license to continue with the installation.
6. On completion, Eclipse prompts you to restart. Restart Eclipse.

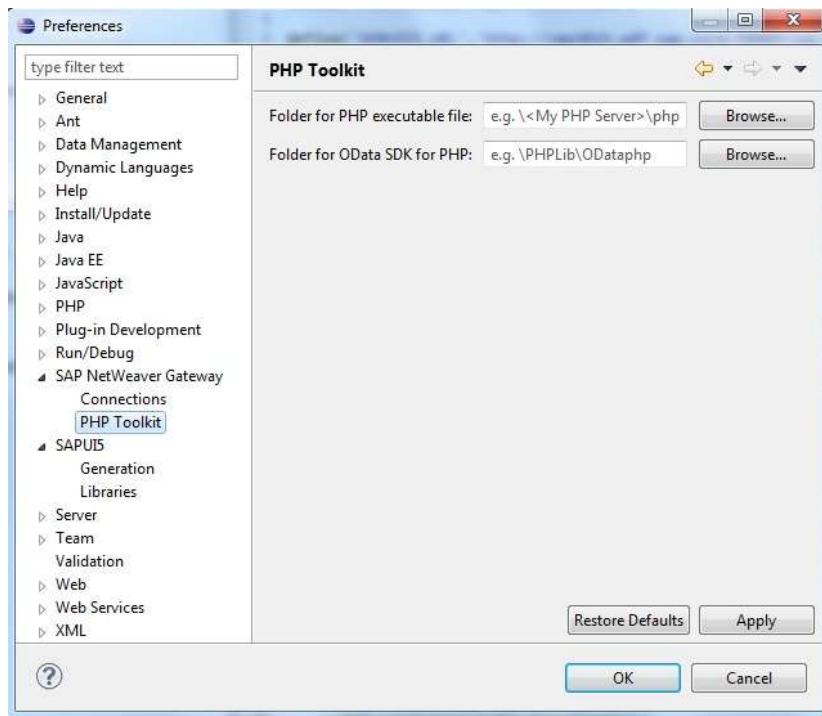
Post installation: Configuring the PHP Toolkit

After installing the PHP Toolkit, you must configure the OData SDK for PHP, and the PHP executable file for the framework.


Make sure that you have installed and configured the OData SDK for PHP in your PHP server.

You configure the installation by specifying the path to the installed PHP executable file, and the OData SDK for PHP.

1. From the Eclipse main menu, go to Window → Preferences → SAP NetWeaver Gateway → PHP Toolkit, and enter the full path for the PHP executable file. For example, C:\Program Files\PHP\



2. Set the path of the OData SDK for PHP.

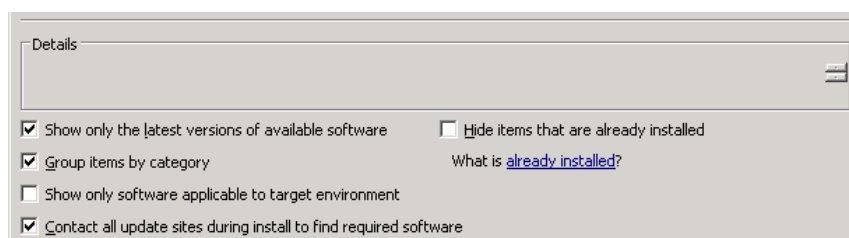
 **Caution:** Copy only the files from the "framework" folder into the "odataphp" folder. Do not copy the entire framework folder into the odataphp folder.

Installing the PHP Development Tool (PDT) SDK Feature Plugin

 We recommend that you install the PHP Development Tool (PDT) SDK Feature Plugin for best presentation of the generated code for PHP.

To install the PHP Development Tool (PDT) SDK Feature Plugin:

1. From the Eclipse main menu, go to **Help** → **Install New Software** → **What is Already Installed?**



2. Select the update site of your Eclipse version and release, search for **PDT** and then select **PHP Development Tools** from the list. Proceed with the installation.

For more information about the PDT plug-in, go to www.eclipse.org/projects/project.php?id=tools.pdt

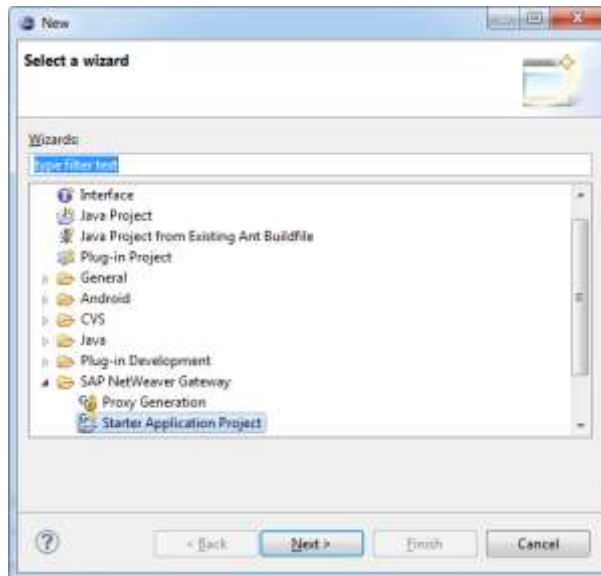
Generating a PHP Starter Application

You can create a PHP starter application that calls SAP services in SAP NetWeaver Gateway.

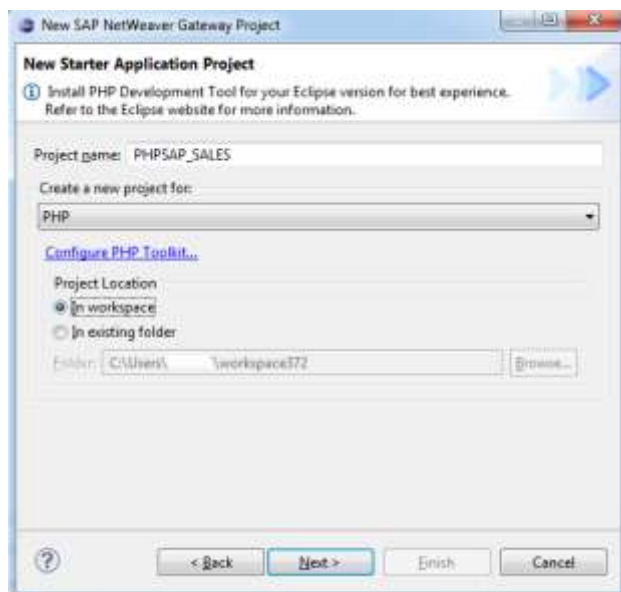
Using the project wizard, you generate code for PHP into the new PHP project that you specify.

To create your PHP starter application:

1. From the File menu, choose **New** → **Other** → **SAP NetWeaver Gateway**, or from the keyboard press **CTRL+N**, and select **Starter Application Project**. The New wizard dialog displays.




2. Enter a new name for your project.
The project and the relevant resources will be created for only your application.
3. From **Create new project for**, select **PHP**, and then specify the location for creating a folder for the project.



A pane for specifying the details of the application as required by the selected target extension opens.

The following are the options:

- In workspace:
A new folder is created for the project and the PHP application in the default folder location in Eclipse.
- In existing folder:
Choose Browse to specify the existing folder in which you want to locate the project and the PHP application.

 Note: Specify the appropriate folder in your existing PHP server as your project location in order to run and test your application in Eclipse using the PDT plug-in.

4. Choose **Next**. The page for the SAP templates for the supported extension displays.
List/Details Application template is the only template available for your PHP application.
5. Choose **Next**. The page for specifying the SAP service you want your application to call displays.




6. Select Remote location, to connect to your SAP NetWeaver Gateway server within the network.
7. Click Catalog to search for the specific SAP service. The exploration dialog opens for you to search for the service.

When you click Catalog, without providing a specific URL, the specified default SAP NetWeaver Gateway is automatically selected.

If you have not saved the password of the default SAP NetWeaver Gateway, a popup displays requesting the password.

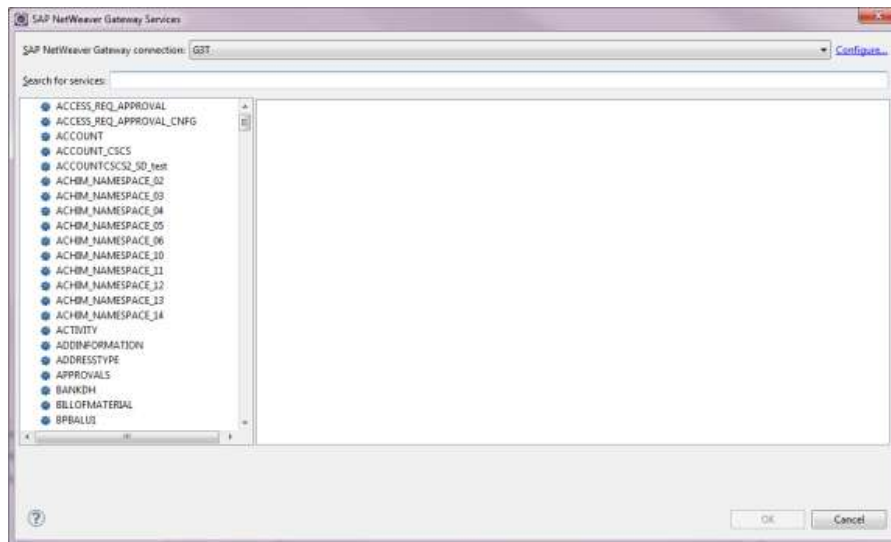
Alternatively, select File system, and click Browse to specify the paths to both the service metadata document for the specific service, and the service document in your file system.

Click Validate to verify that the service documents are properly formed XML files for OData compliant services.

 Note: If the specified URL of the service metadata comes from the Search Console, the Validate button is not available, as both the connection and the metadata document is verified.

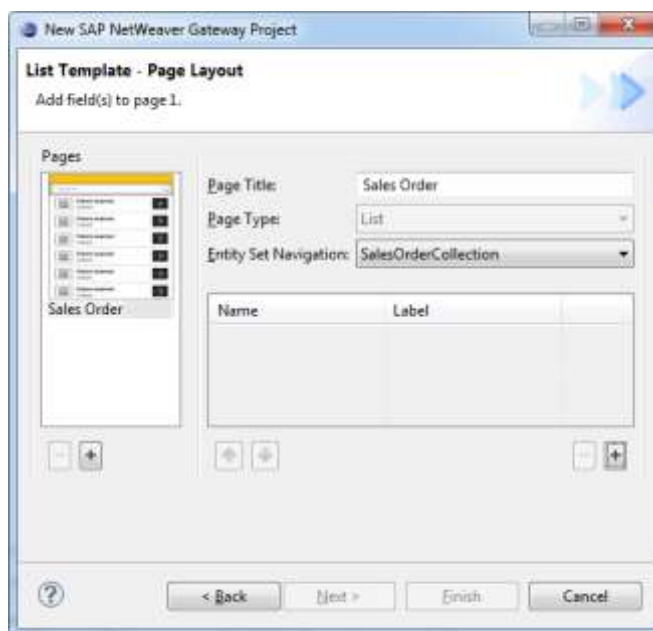
The Validate button is available only if you have manually entered the URL, or selected the metadata file from the file system.

8. Expand the service group and select the service you want. For example, BANK_DETAILS, and choose **Next**. The List Template-Page Layout screen displays.



9. Enter the following for the default list page:

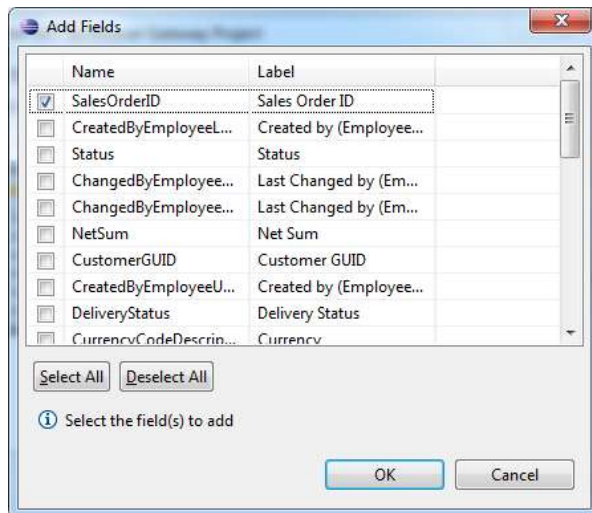
- Title: Enter a title for the page.
- Type: Select the page type. There are two page types. These are list and details page.
- Entity Set Navigation: Select an object of the SAP service from which you can choose the fields and properties to add to the page.



10. Add fields to the default page or add extra pages.

To add fields to a page, click the plus sign (+) under the display area for fields. The Add Fields dialogue displays.

You can remove selected fields by using the minus sign (-).

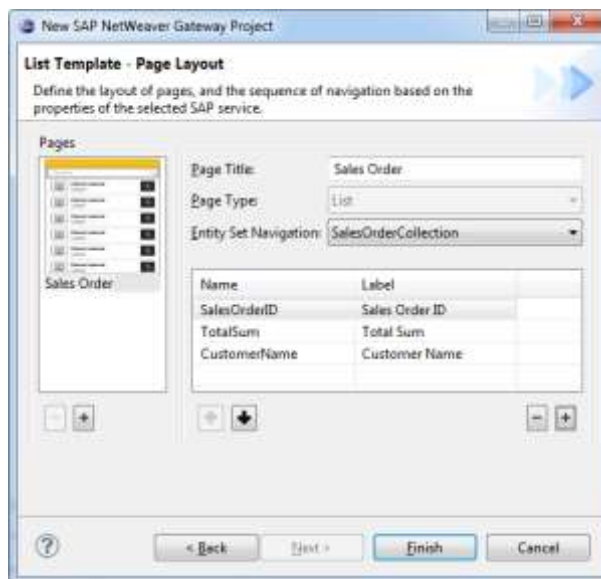


- Select the fields to add and choose **OK**. When you choose **Select All**, all the fields will be selected.
- To remove the selection for a field to be added, click the selected field. When you want to remove the selection for all fields, choose **Deselect All**.

Later, you can arrange the sequence in which the fields will be presented at runtime using the arrow down (↓) and the arrow up (↑).

 **Note:** For each page that you add, specify the fields you want to make available.

11. Choose **Finish**.



Code is generated for each page you defined page in the wizard. To add more pages and fields to the pages, see the section that follows.

Extending the Generated PHP Starter Application

You can modify and extend the generated code for the PHP starter application to suit your needs.

Your PHP starter application files based on the List Application template are created and located in the specified Eclipse project and workspace. The number of files created depends on the number of pages you defined in the wizard.

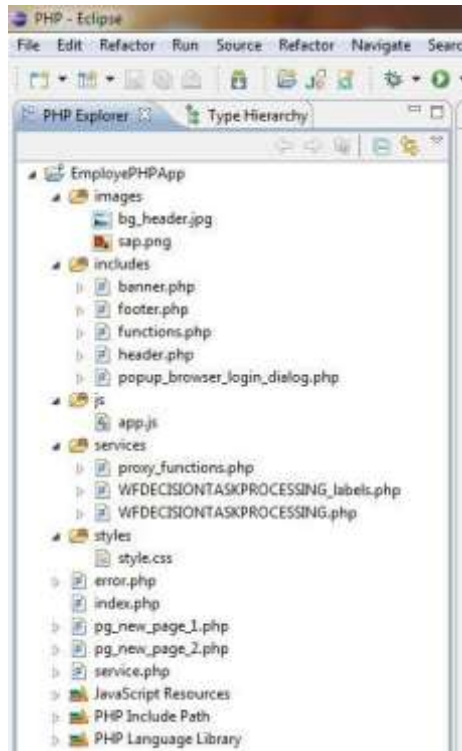
An example of the contents generated using the List Application template is provided in the section below.

You can run and test the generated application immediately.

The generated project is a PHP project in Eclipse, which can be presented in any perspective.

We recommend to use the PHP perspective of the PDT. Find more information about the PDT plug-in at: <http://www.eclipse.org/projects/project.php?id=tools.pdt>.

Below is an example of a generated project for the PHP toolkit.



The generated project consists of the following:

The application pages (PHP files with the prefix pg_*.php) display the information received from the SAP NetWeaver Gateway service, according to the user selections specified in the wizard.

- index.php
For redirection to the first page of the application.
- service.php
For initializing the SAP NetWeaver Gateway service proxy class.
- services folder
Generated proxy class and property labels array, for the chosen SAP NetWeaver Gateway service. In addition, there are several utility functions in the file, proxy_functions.php
- error.php
An error page of the application, used for displaying error messages.
- includes folder
Contain helper PHP files which are included from the application pages. The following are the files:
- banner.php
Application name and logo image.
- footer.php
HTML footer.

- `functions.php`
A set of PHP functions for authentication, error handling and entry ID extraction.
- `header.php`
HTML header include file.
- `popup_browser_login_dialog.php`
Returns 401 and authentication response to the browser.

Application resources consist of the following folders:

- Images folder: Contains images used in the generated application
- Styles folder: Contains the file, `styles.css`, a style sheet with user interface styles and themes.
- JS folder: Contains the file, `app.js`, a JavaScript file for filtering and sorting of HTML table.

Eclipse project files for the following: `.buildpath`, `.project`, and `.setting` folder.

Styling and Runtime User Interface

The application maintains clear separation between the business logic and the runtime user interface themes and styles, for example, *PHP/HTML/CSS/JavaScript*.

You can modify the following:

- A stylesheet is provided and you can add your own stylesheet file. The `style.css` file contains an out-of-the-box theme for your application with predefined colors, images and positioning for the HTML controls.
- The `banner.php` file contains the preset banner area for the application which you can edit as needed.

The application pages display the SAP service data using a simple HTML table, which you can edit if necessary.

HTML Header and Body of the Application Pages

The `header.php` file contains all the necessary elements to construct the HTML page metadata along with the HTML header.

The `$title` variable is set by each page before including the `header.php` file. This allows you to set different titles for different pages.

In addition, the `header.php` file includes references to the required JavaScript and CSS files along with the standard `<meta>` tag for HTML pages.

The `footer.php` file contains closing tag element of `<body>`. You can add your own custom footer if needed.

Client-Side Functionality

The Javascript file, `app.js`, is responsible for the client-side code that supports filtering and sorting of the list in an HTML table.

Filtering

Filtering is enabled and all the rows are visible when the page is loaded.

When you enter a filtering value in the filter textbox, only the rows that match the specified value display in the table. The other rows are not displayed.

In the filter textbox, press the `ESC` key to clear its content and to display the content of all rows again.

When you enter any other value in the filter textbox, the filter method is executed, passing the value to be filtered.

The filter method uses the `text().search()` jQuery method on the row content to find matches and then adds or removes the 'visible' class accordingly.

Sorting

For sorting support, when you click a table column header, the code loops through all the cells of the specific column and then sorts them using the `sort()` jQuery method.

A sort direction is also kept by setting a CSS class accordingly ('*sorted-sac*'/'*sorted-desc*').

Note that the navigation link to the next page is excluded from filtering and sorting by identifying the column header identifier: `<th id="ListNavHeader">`, and the CSS class for table value: `<td class="details">`, used in the application list pages.

The `app.js` file also handles the 'hovering' of each row by adding a CSS class to color the background of the row.

The client-side functionality can be disabled in the application by adding comments to the JavaScript files references in the `header.php` file.

Service Proxy

The services folder contains a proxy class for the chosen SAP NetWeaver Gateway service, as well as a file containing the service property labels array.

The proxy code is generated by the OData SDK tool for PHP. More information regarding the SDK can be found at: odataphp.codeplex.com

The `service.php` file creates a new instance of this proxy, passing it the service URL.

It then adds the credentials which were passed by the browser. For more information, see the section, *Secure PHP Application*.

Finally it adds a callback function which is called before any HTTP request is sent by the service proxy. This method adds the request for HTTP headers required by the SAP NetWeaver Gateway server, as SAP client.

The `service.php` file defines the service URL and SAP client as constants. Later, you can configure the URL address and the SAP client of the SAP service without modifying the application pages.

To use HTTP proxy to access the service, remove the comments in the appropriate code in the `service.php` file, and define the HTTP proxy server details there.

You can use the `$proxy` variable from any application page. Make sure that you include the `service.php` file in such pages. The `$proxy` variable is used in the generated application pages to access the service data and to fill the tables in the page.

You can use the proxy for additional service calls. For more information about the OData proxy, refer to the user guide located in the `doc` folder of the installed OData SDK for PHP.

Generating a Proxy for a PHP Application

You can use the Proxy Generation wizard in the framework to generate PHP proxy classes for the specified OData compliant SAP service.


The wizard creates a service folder containing the PHP files that it had created for the selected SAP service, as well as a file containing the service property labels array.

The proxy code is generated by the OData SDK tool for PHP. More information regarding the SDK can be found at: odataphp.codeplex.com.


You can call the generated proxy files from your PHP application to interface with the selected SAP service through the specified SAP NetWeaver Gateway server.

Note that you have to create your own user interface for rendering your application.

You can generate a starter PHP application to see an example of the code for making calls to the generated code for an SAP service in the PHP environment.

-  The following is an example of a class that calls the generated classes for the SAP service, *RMTSAMPLEFLIGHTService*.


The code sample used in this example is available on the SAP Developer Network (SDN) at: [wiki.sdn.sap.com/wiki/display/Snippets/SAP NetWeaver Gateway-Code Snippets](http://wiki.sdn.sap.com/wiki/display/Snippets/SAP+NetWeaver+Gateway+Code+Snippets)

-  Each line in the example has comments to help you understand the expected process:

- Initialize the service through its URL.

```
  
<?php  
require_once "RMTSAMPLEFLIGHT.php";  
$proxy = new RMTSAMPLEFLIGHT  
( 'http://<Gateway_Host>:<Gateway_Port>/sap/opu/sdata/IWFND/RMTSAMPLEFLIGHT' );  
$proxy->Credential = new WindowsCredential( '<Username>', '<Password>' );
```

- Read example: Get carrier by carrier ID with an authenticated X.509 client certificate. Register a callback function to add the client certificate to the request.

```
  
$proxy->OnBeforeRequest("DoBeforeRequest", null);
```

- Get carrier by carrier ID, for example, American Airlines.

```
  
try  
{  
    $carrier_id = 'AA';  
    $query = $proxy->CarrierCollection()->filter("carrid eq '$carrier_id'");  
    $carriers = $query->Execute()->Result;  
    echo "Carrier ID: " . $carriers[0]->carrid . "<br/>";  
    echo "Carrier Name: " . $carriers[0]->CARRNAME . "<br/>";  
    echo "Carrier Code: " . $carriers[0]->CURRCODE . "<br/>";  
    echo "Carrier URL: " . $carriers[0]->URL . "<br/>";  
}
```

- Error handling example.

```

catch(DataServiceRequestException $exception)
{
    echo $exception->Response->getError();
}

```

- Callback function to add the client certificate to the request.

```

function DoBeforeRequest($HttpRequest)
{
    $refProp = new ReflectionProperty('HttpRequest','_curlHandle');
    $refProp->setAccessible(true);
    $curlReq = $refProp->getValue($HttpRequest);
    curl_setopt($curlReq, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curlReq, CURLOPT_FOLLOWLOCATION, true);
    curl_setopt($curlReq, CURLOPT_SSL_VERIFYHOST, 1);
    curl_setopt($curlReq, CURLOPT_SSL_VERIFYPEER, true);
    curl_setopt($curlReq, CURLOPT_CAINFO,
        '<Gateway_CA_Certificate_Path>.cer');
    curl_setopt($curlReq, CURLOPT_SSLCERT, '<Client_Certificate_Path>.pem');
    $refProp->setAccessible(false);
}

```

- Create example: adding new booking information for a flight for a specific flight of the carrier. In addition, show all errors, except for notices and coding standards warnings the OData library.

You create a booking object with all the properties filled.

```

error_reporting(E_ALL & ~E_NOTICE);
try
{
    $carrid = 'AA';
    $connid = '0017';
    $fldate = '2011-12-21T00:00:00';
    $bookid = '';
    $booking = Booking::CreateBooking($carrid, $connid, $fldate, $bookid);
    $booking->CUSTOMID = '00004617';
    $booking->CUSTTYPE = 'P';
    $booking->WUNIT = 'KG';
    $booking->LUGGWEIGHT = '14.4000';
    $booking->CLASS = 'Y';
    $booking->FORCURAM = '879.82';
    $booking->FORCURKEY = 'USD';
    $booking->LOCCURAM = '803.58';
}

```

```

$booking->LOCCURKEY = 'USD';
$booking->ORDER_DATE = '2011-05-22T00:00:00';
$booking->COUNTER = '00000000';
$booking->AGENCYNUM = '00000325';
$booking->PASSNAME = 'Joe Smith';
$booking->PASSFORM = '1234567';
$booking->PASSBIRTH = '1990-10-10T00:00:00';

```


- Add the new booking object to the booking collection.

```

<>
$proxy->AddToBookingCollection($booking);

```

- Add custom headers to the POST request and execute the request.

 **Note:** The service is aware of these changes when you call the method, *SaveChanges*.

```

<>
$proxy->SetSaveChangesOptions(SaveChangesOptions::None);
$proxy->SetEntityHeaders($booking, array('X-Requested-With' =>
'XMLHttpRequest'));
$proxy->SaveChanges();

```

- Function import: Example for using the flight service specific method to get flights between specific dates and destination. Obtaining available flights.


```

<>
try
{
$proxy->addHeader('X-Requested-With', 'XMLHttpRequest');
$city_from = 'NEW YORK';
$city_to = 'SAN FRANCISCO';
$from_date = '20110105';
$to_date = '20110728';
$flights = $proxy-
>Execute("GetAvailableFlights?cityfrom=$city_from&cityto=$city_to
&fromdate=$from_date&todate=$to_date")->Result;
foreach ($flights as $flight)
{
echo "Flight Carrier: " . $flight->carrid . "<br/>";
echo "Flight Price: " . $flight->PRICE . "<br/>";
echo "Flight Max. Seats: " . $flight->SEATSMAX . "<br/><br/>";
}
}
}

```


- Below is an example code for getting media links, based on the Gateway service, *WFODCPROCESSING*, which is different from the service used in the example above.

It obtains a list of attachments for a specific workflow task from the Gateway server.


```

$response = $proxy->Execute("WorkflowTaskCollection(task_id)
/Attachments");
$attachmentList = $response->Result;
```

- Obtains a specific attachment.


```

$attachment = $attachmentList[0];
```


- Gets the mime type of the attachment.

```

$mimeType = $attachment->mime_type;
```

- Obtains the byte stream of the media resource

```

$proxy->GetReadStream($attachment,null)->getStream();
```

- Add your own code to use the media resource to suit your needs.
For example, if the mime type is a JPEG, you can display it as an image.

```


```

Running your PHP Application

You can run the PHP application immediately or edit the code in the generated class, and perform various tests for it.

Run and test the generated PHP application in Eclipse using the PDT plug-in. To do so, make sure that the project location (specified in the first page of the wizard) is the appropriate folder in the PHP server.

Secure PHP Application

The generated PHP application supports basic authentication that uses the Browser pop-up dialog for user credentials.

Each application page includes the *popup_browser_login_dialog.php* file, which checks if the request from the browser contained basic authentication credentials (checks for the username). If not, it calls the **authenticate()** method from the *functions.php* file.

The method tells the browser to open the basic login dialog in order to allow users to enter their username and password. The username and password are not stored in the server.


To disable the basic login dialog pop-up, add comments to the call to the **authenticate()** method, in the *popup_browser_login_dialog.php* file.

To enable anonymous access to the SAP NetWeaver Gateway service, add comments to the call to the proxy credential settings (with the values entered in the login dialog), in the *service.php* file.

PHP Application Support for Cross Site Request Forgery Protection

Most OData compliant SAP services are protected against Cross-Site Request Forgery (CSRF) attacks. This protection mechanism appends challenge tokens to each request and associates them with each user's session. Each token is unique per user session.

You must enable your PHP application to make requests that include a challenge token and ensure that the requests from your application are not coming from another source other than the user.

-  **Note:** All SAP services that are compatible with OData library version 1.0 are protected and require XCSRF tokens. Some services that are compatible with OData library 0.2 also require these tokens.

Services that contain Create, Update, and Delete (CUD) operations (POST, PUT, and DELETE HTTP methods) require XCSRF tokens.

To authenticate the requests from these services, do the following:

1. Open the file, **service.php**, and locate the event, **doBeforeRequest**.
2. Before the **doBeforeRequest** event, manually add calls to one of the following methods. These methods are in the file, **proxy_functions.php**:

- `add_xcsrf_headers`


The method, **add_xcsrf_headers**, will add XCSRF headers to the given `HttpRequest` object (representing an HTTP POST, PUT, and DELETE request before sending the request).

Using the URL, it will retrieve the required XCSRF data from the SAP NetWeaver Gateway host. This method creates new XCSRF header data for each call.

- `get_xcsrf_data`

The method, **get_xcsrf_data**, obtains the XCSRF data (token and cookie) in order to set the appropriate XCSRF headers required for any HTTP POST, PUT, and DELETE request sent to the SAP NetWeaver Gateway host.

It uses the URL and user credentials to retrieve the required XCSRF data from the SAP NetWeaver Gateway host. This method creates a call for XCSRF data and saves it.

-  **For example**

```

function doBeforeRequest($HttpRequest)
{
    addHeadersBeforeRequest($HttpRequest);
    ....;
}

function addHeadersBeforeRequest($HttpRequest)
{
    if(SAP_CLIENT != '')
    $HttpRequest->Headers->Add('sap-client', SAP_CLIENT);
    $HttpRequest->Headers->Add('X-Requested-With', 'XMLHttpRequest');
    if($HttpRequest->getMethod() != HttpVerb::GET)
    add_xcsrf_headers(SERVICE_URL, $HttpRequest);
}
```

PHP Application Support for X.509 Client Certificate Authentication

In a productive environment, configure the PHP application to access the SAP NetWeaver Gateway server using X.509 client certificate authentication over SSL.

The following is an overview of the tasks for enabling X.509 support for your application:

1. Authenticate the user in the PHP application server, and generate an appropriate X.509 client certificate for the user in the application context.

The certificate should be signed by a certificate authority, CA, trusted by the SAP NetWeaver Gateway server.

For more information, see *Generate and Sign X.509 Client Certificates*.
2. Save the generated client certificate in a secured store within the PHP application server. Contact your system administrator for assistance.
3. Enable SSL validation in the SAP NetWeaver Gateway server for the requests sent from the PHP application.
For more information, see *Sending the Service Request with the Client Certificate*.
4. Attach the generated client certificate to the SAP NetWeaver Gateway server requests sent from the PHP application on behalf of the specific user.
For more information, see *Sending the Service Request with the Client Certificate*.

Sending the Service Request with the Client Certificate

In order to send service requests using X.509 client certificate authentication, attach the certificate to any request you send from the service proxy.

In addition, enable SSL server validation for all the requests you send. To do so, change the file *service.php* as follows:

1. Remove the comments before the ***addClientCertificate()*** method code and the initiate a call to it in the ***doBeforeRequest()*** method.
2. Replace the placeholder, ***<Gateway_CA_Certificate_Path>***, in the ***addClientCertificate()*** method code, with the path of the root CA certificate trusted by the SAP NetWeaver Gateway server saved locally to the PHP application server. Make sure that the certificate is saved in Base64-encoded format.
3. Replace the placeholder, ***<Client_Certificate_Path>***, in the ***addClientCertificate()*** method code with the path of the stored X.509 client certificate generated for the user. Make sure the certificate is saved in PEM format.
4. Comment the call to the proxy credential settings (with the values entered in the login dialog).

You can disable the basic login dialog pop-up used for authentication in the PHP application server.

To do so, comment the call to the ***authenticate()*** method in the ***popup_browser_login_dialog.php*** file.

Generate and Sign X.509 Client Certificates

You can generate and sign X.509 client certificates dynamically (for a specific user using OpenSSL or other APIs that enable creating and signing of certificates in PHP).

For more information about OpenSSL, see:

www.php.net/manual/en/function.openssl-csr-new.php

Creating Your Custom PHP Template

You can create a new PHP template for use in the framework. The new template is based on the appropriate pattern and environment of the PHP toolkit in the framework.

Requirements

Make sure that you have:

- Added the correct plug-in dependency for the PHP Toolkit, ***com.sap.iw.gw.oc.eclipse.php***.

For more information, see Adding the Required Dependency Plug-in.

- Specified the correct environment identifier attribute for the PHP Toolkit, ***com.sap.iw.gw.oc.eclipse.environment.php***.

For more information, see Creating the New Template Class.

The created template class receives the PHP environment from the framework. This environment class implements the *com.sap.iw.gw.oc.eclipse.environment.php.IPHPEnvironment* interface, and exposes the PHP environment parameters that have been defined in the environment implementation (as the selected project name and location).

In addition, the environment class provides some methods for creating an empty PHP project, into which the framework generates proxy files and uses the PHP perspective.

The following is an example code (using the Exploration Pattern Identifier *com.sap.iw.gw.oc.eclipse.pattern.exploration*) for implementing the class methods for your new template.

The template will generate a new PHP project including proxy files of a selected service, and a reference to the OData for PHP library:

```
private IPHPEnvironment phpEnvironment;
private IServiceModel serviceModel;

@Override
{

@Override
public void onFinish(IProgressMonitor monitor) throws TemplateException {
    monitor.beginTask("Generating Custom Application with PHP proxy ...", 5);
    this.serviceModel = (IServiceModel) this.context.get(TemplateConstants.PATTERN);
    this.phpEnvironment = (IPHPEnvironment)
        this.context.get(TemplateConstants.ENVIRONMENT);
    String projectName = this.phpEnvironment.getProjectName();
```

- Create an empty project.

```
String projectHomePath = null;
try {
    projectHomePath = this.phpEnvironment.createEmptyPHPProject();
}
catch (EnvironmentException e) {
    throw new TemplateException(e);
}
monitor.worked(1);
```

- Generate the proxy files for the service, and the property labels resource files, and add them to the empty project.

```
try {
    this.phpEnvironment.generatePHPProxy(this.serviceModel, projectHomePath);
}
catch (EnvironmentException e) {
    throw new TemplateException(e);
}
monitor.worked(2);
```

- Refresh the created project in Eclipse with the added files.

```
try {
    ProjectUtils.refreshProject(projectName);
}
catch (CoreException e)
{
    throw new TemplateException(e);
}
monitor.worked(1);
```

- Switch to the PHP perspective.

```
this.phpEnvironment.openPHPPerspective();
monitor.worked(1);
System.out.println("Successfully generated proxy into project " + projectName);
monitor.done();
}
```

Using the Android Toolkit

The SAP NetWeaver Gateway plug-in for Eclipse – Android Toolkit consists of an environment, a pattern and templates, suitable for developing SAP solutions for use in the Android environment.

Software Prerequisites

Ensure that the following prerequisites are met:

- You have downloaded and installed the SAP NetWeaver Gateway plug-in for Eclipse Framework.

For SAP NetWeaver Gateway plug-in for Eclipse – Android Toolkit; you require the following:

- Android SDK Tools (Android API level 13).

The SDK starter package is not a full development environment, it includes only the core SDK Tools, which you can use to download the rest of the SDK packages (such as the latest Android platform).

SDK Tools is a downloadable component for the Android SDK. It includes the complete set of development and debugging tools for the Android SDK.

- SDK Manager: SDK Tools r16:
http://dl.google.com/android/android-sdk_r16-windows.zip
- Download and extract to local file system
- Platforms: Start *SDK Manager.exe* and download Android SDK Platform for 3.2 (API 13)

For more information see, refer to developer.android.com/sdk/index.html

- Android Development Tools (ADT) plug-in for Eclipse (ADT version 16 and higher).

Download and extract the ADT plug-in to your local file system. For more information, see, <http://dl.google.com/android/ADT-16.0.1.zip>

In Eclipse, choose **Help** → **Install New Software** → **Add** to add the path to the local updatesite and install it from there.

After installation, the ADT Plugin prompts you for Android SDK location, enter the path to the SDK Platform Tools you have already installed.


For more information, refer to developer.android.com/sdk/eclipse-adt.html

- SAP OData Mobile Client SDK for Android. For more information, refer to [Sybase InfoCenter](#).

Download the SAP OData Mobile Client SDK for Android from the SAP Developers Network site (SDN) at:
<http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/webcontent/uuid/20ece4e6-1b5a-2f10-8ba2-8c99ef25b9d9>

The zip contains the JAR files which you must extract to a location on your file system. These are required by the SAP NetWeaver Gateway plug-in for Eclipse.

Installing the Android Toolkit

-  **Note:** Make sure that you have met all the software prerequisites specified above. You cannot proceed with the installation if one or more of the prerequisites have not been met.

Browse the installation package in your local file system to locate the target environment as follows:

1. Start Eclipse, and from the main menu choose **Help → Install new software**. The Install dialog displays.
2. Click **Add**, and then choose **Archive**.
3. Select the zip file, **SAPNetWeaverGatewayPluginForEclipse.zip**, from the file system location where it was downloaded.
4. Choose **OK**, and select the option to install SAP NetWeaver Gateway plug-in for Eclipse – Android Toolkit.
5. Accept the license to continue with the installation.
6. On completion, Eclipse prompts you to restart. Restart Eclipse.

Configuring the Android Toolkit

After installing the Android Toolkit, you must configure the SAP OData Mobile Client SDK for the framework.

You configure the SAP OData Mobile Client SDK by specifying the path to the files for the SAP OData Mobile Client SDK.

1. From the Eclipse main menu, go to **Window → Preferences → SAP NetWeaver Gateway → Android Toolkit**.
2. Enter the full path to the SAP OData Mobile Client SDK. For example, C:\SUP\AndroidLibraries

Generating an Android Starter Application

You can create one of the following starter applications for Android, to call SAP services in SAP NetWeaver Gateway:

- A List/Details Starter Application.

Creates a list and details starter application for Android using any properly formed SAP services that are Odata-compliant.

- A Basic Starter Application.

Generates code for use as an example of how to read properties from one of the addressable collections of the selected Gateway service.

It is a starter application structure with entry points for custom development.

When you choose this template, the wizard generates the folder structure, JAR files and class structure required for an application consuming SAP NetWeaver Gateway, including code sample for using the generated proxy with a simple UI and an option to generate JUnit project with code sample for CRUD operations.

- A Workflow Starter Application.

Creates a starter application for Android using only Workflow services from the Gateway.

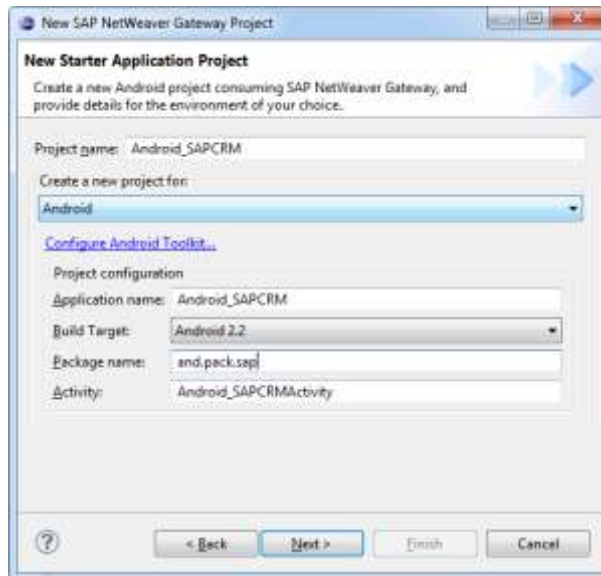
The template showcases a best-practice for creating a business-scenario application.

The generated project contains a proxy for the service you selected in the wizard.

To create your Android starter application:


1. From the File menu, choose **New → Other → SAP NetWeaver Gateway**, or from the keyboard press **CTRL+N**, and select Starter Application Project. The New wizard dialog displays.

2. Enter a new name for your project. The project and the relevant resources will be created for only your application. For example, SAPAndroid.
3. From **Create new project for**, select **Android**.



A pane for specifying the details of the application as required by the selected target extension opens.

4. Enter the following:
 - **Project Name:**
Specify a name for creating a new Android project for the starter application.
 - **Application Name:**
Specifies the name of the application to render to users.
 - **Package:**
Specify a package name that conforms to the Java package naming format. For example, sap.android.crm
 - **Target Build:**
Specify the Android platform version for the application.
Android 2.2 is the minimum supported Android platform version.
 - **Activity:**
Specifies the Android activity for the application.
5. Choose **Next**. The page for the SAP templates for the supported extension displays.
6. Select the template you want to use. The following are the templates:
 - List/Details Application template is provided for your application.

 **Note:** You can create and add your custom templates to the templates list for the Android environment.
 - Basic Application template that generates code as an example of reading properties from one of the addressable collections of the selected Gateway service.
 - Workflow Application template is for your SAP Workflow application.
7. Choose **Next**. The page for specifying the SAP service you want your application to call displays.




8. Select **Remote location**, to connect to your SAP NetWeaver Gateway server within the network.
9. Click **Catalog** to search for the specific SAP service. The exploration dialog opens for you to search for the service.

When you click Catalog, without providing a specific URL, the specified default SAP NetWeaver Gateway is automatically selected.

If you have not saved the password of the default SAP NetWeaver Gateway, a popup displays requesting the password.

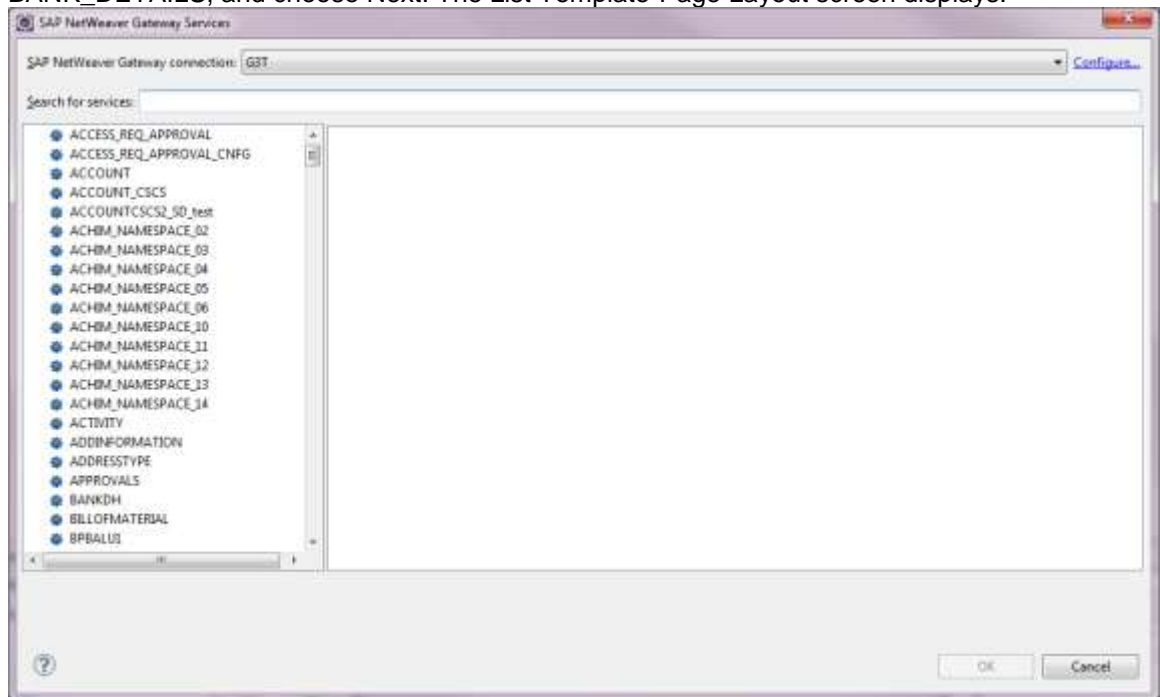
Alternatively, select **File system**, and click **Browse** to specify the paths to both the service metadata document for the specific service, and the service document in your file system.

Click **Validate** to verify that the service documents are properly formed XML files for OData compliant services.

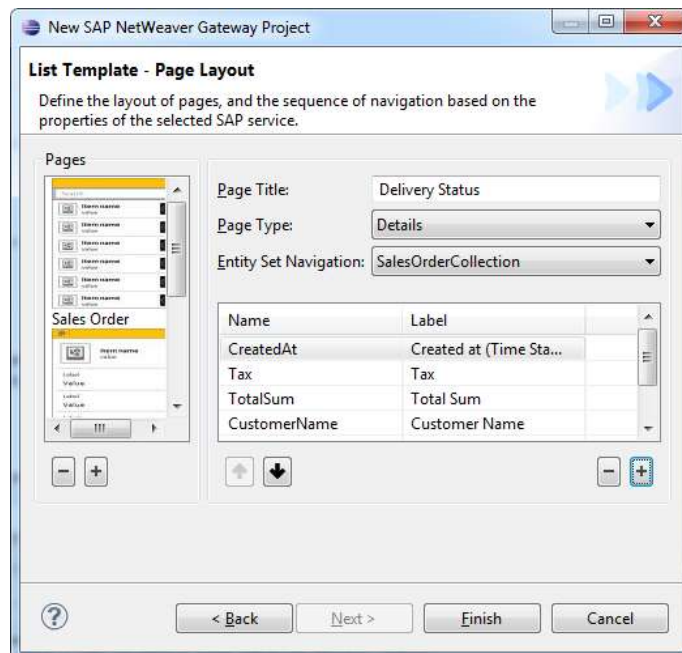
 **Note:** If the specified URL of the service metadata comes from the Search Console, the Validate button is not available, as both the connection and the metadata document is verified.

The Validate button is available only if you have manually entered the URL, or selected the metadata file from the file system.

10. Expand the service group and select the service you want. For example, **BANK_DETAILS**, and choose Next. The List Template-Page Layout screen displays.



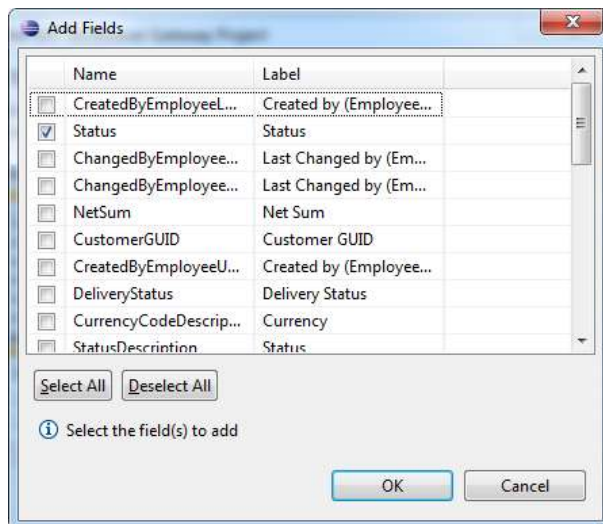
11. Enter the following for the default list page:
 - Title: Enter a title for the page.
 - Type: Select the page type. There are two page types. These are list and details page.
 - Entity Set Navigation: Select an object of the SAP service from which you can choose the fields and properties to add to the page.



12. Add fields to the default page or add extra pages.

To add fields to a page, click the plus sign (+) under the display area for fields. The Add Fields dialogue displays.

You can remove selected fields using the minus sign (-).



13. Select the fields to add and choose OK. When you choose Select All, all the fields will be selected.
14. To remove the selection for a field to be added, click the selected field to remove the check mark. When you want to remove the selection for all fields, choose Deselect All.

Later, you can arrange the sequence in which the fields will be presented using the arrow down (↓) and the arrow up (↑).

For each page you add, specify the fields you want to make available.

Note: If you generate code for an application based on the Basic Application template, both the Finish and the Next buttons are available.

15. Choose **Next** to select or remove the selection to generate an Android JUnit project with sample code.
16. Choose **Finish**.

Code is generated for each page you defined page in the wizard.

Android Junit Project

A sample class is created in the JUNIT project under the specified package name for the basic application. The test class file is called, *DemoTest*, Appended to the end of this package name is the term, *test*.

Note: Available only when creating an Android application based on the Basic Application template of the Android Toolkit.

The generated sample class contains one or more methods for the following CRUD operations: ***testGet***, ***testUpdate***, ***testDelete*** and ***testCreate***.

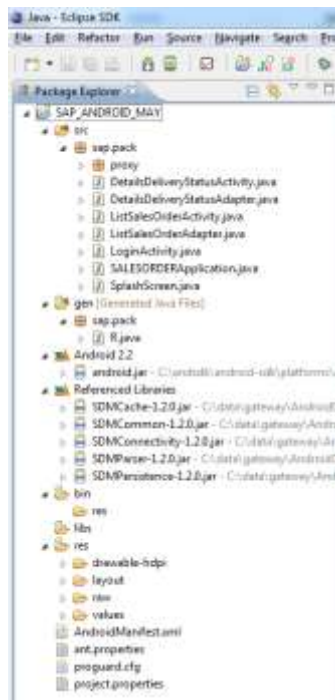
Each specific test type is created only if the operation is supported by the service.

Extending the Generated Android Starter Application

You can modify and extend the generated code for the Android starter application to suit your needs.

Your Android starter application files based on the List/Details Application template are created and located in the specified Android project in the Eclipse workspace. The number of files created depends on the number of pages you defined in the wizard.

Below is an example of the contents generated using the List/Details Application template:



You can run and test the generated application immediately. The generated project is an Android project in Eclipse.

Extending the Generated Android Basic Application

You can modify and extend the generated code for the Android Basic application to suit your needs.

The generated code provides an example for reading properties from one of the addressable collections of the selected Gateway service.

Your Android application files based on the Basic Application template are created and located in the specified Android project in the Eclipse workspace. The number of files created depends on the number of pages you defined in the wizard.

It includes code sample for using the generated proxy for the service in a simply rendered user interface.

Also, the generated Android application contains code for using the SUP client library.

To run the generated Android Basic application:

1. From the newly create Android project, go to the folder, src, expand your package and open the activity class, for example, <project_name>Activity.java
2. Go to the following line and replace the placeholder; username and password of the service with the actual logon credentials to your SAP system:

```
public class AndroidCustomActivity extends Activity
{
    private String userName = "<userName>";
    private String password = "<password>";
}
```

3. Run the application.

Extending the Generated Workflow Android Application

The generated Workflow Android application enables users to view and manage their Workflow tasks using a Workflow service in the SAP NetWeaver Gateway server.

Depending on the Workflow service, your application is composed of 4 components:

- Splash Screen
- Login screen
- Workflow Task List
- Workflow Task details

Splash Screen

The splash screen is used to notify the user that, the application is in the process of loading. It disappears when the application's main window displays.

The splashscreen logic is handled by the code in the generated project within the SplashScreen.java class.


Login Screen

The login screen handles the user authentication with the SAP NetWeaver Gateway service. Users must provide a user name and a password.

The login and authentication logic is handled by the code of the login method in the LoginActivity.java class.

Workflow Task List

The main screen of the application displays the Workflow task list and access to the details of each task.

 Note: The task list is obtained from the SAP NetWeaver Gateway the first time the application is loaded.

For each task, the following details are displayed:

- Subject
- Creator
- Creation date
- Priority

Task priorities are received from the service as numbers and are mapped to strings using the following mapping:

1- Highest	6- Low
2- Very High	7- Lower
3- Higher	8- Very Low
4- High	9- Lowest
5- Medium	

The task priorities are also marked with colors: high priorities (1-4) in red, medium priority (5) in black, and low priorities (6-9) in grey.

Task Filter

The tasks can be filtered by task name which usually represents the type of the task involved (Leave Request, Purchase Order, and many more).

To filter the tasks, choose **Filter** and select the desired task name, or select **All** to view all tasks.


The filtered task list displays according to your selection.

Choose **Cancel** to exit the filter view without performing any change in the displayed list.

Task Search

In the task list view, you can search for tasks according to a search string.


To perform a search:

1. Click the search field at the top of the view.
 2. Type the search item or term.
 3. The search is performed on all presented properties and the results are displayed while typing the search item.
-  Note: Search is performed only on tasks displayed in the list (which may be filtered).

Workflow Task Details

After selecting a task from the task list, a view opens displaying the task details and enabling users to make a decision regarding the task.

The task details displayed include the following:

- Task subject – displayed at the top of the view as the task title
 - Task properties - status, creation date, creator and priority
 - Task extended properties (X-props).
-  Note: Default extended properties are not displayed.
- Links to additional related task details, such as: comments, participants and attachments.

The task details logic is handled by the code in the generated project within the TaskDetailsActivity.java class.

Make Decision

To make a decision regarding the task:

1. Choose **Make Decision**. The list of actions received from the service for the task displays.
2. Choose an action to apply it or choose **Cancel** to return to the task details.
3. After selecting an action, you can add a comment to be sent together with the task. Choose the button with the action name to apply it. Choose **Cancel** to return to the task details without applying the action.

After applying the action, the Details view is once more displayed.

The next time the application loads, this completed task will no longer appear in the task list.

Viewing Task Attachments

The Details view includes a link to the task attachments (if there are any) showing also the number of attachments included.

To view attachments:

1. Choose **Attachments**.

A view showing the file names of the task's attachments is displayed.

2. Choose an attachment.

The attachment opens in the appropriate content viewer according to the file type.

Viewing Task Participants

The Details view includes a link to the task participants (if there are any). Also, it shows the number of participants included.

Choose **Participants**. A view showing the task participant names and types is displayed.

Viewing Task Comments

The Details view includes a link to the task comments (if there are any) showing also the number of comments included.

Choose **Comments**. A view showing a list of task comments including each comment's creator, creation date, and content is displayed.

Viewing Task Description

The Details view includes a link to the task description (if relevant).

Choose Description. A view showing the task description is displayed.

Generating a Proxy for an Android Application

You can create a proxy for an Android application to consume SAP services through your SAP NetWeaver Gateway server.


The proxy provides the implementation interfaces for the SAP service you choose to use. You must create your own application to interface with the generated proxy.

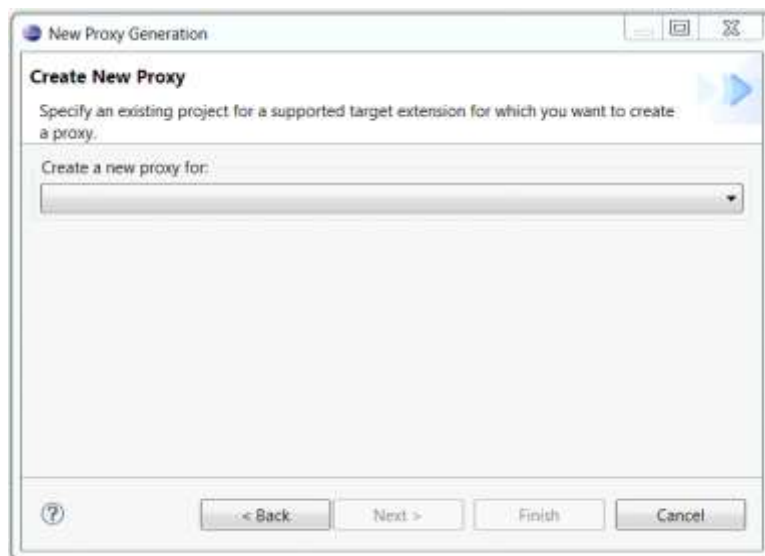
Requirements

Make sure that the selected project is an Android project into which you generate the proxy.

You get a warning message when you attempt to generate code for an environment that is different from the target environment of the selected project.

To create your proxy:

1. From the File menu, choose **New** → **Other** → **SAP NetWeaver Gateway**, or from the keyboard press **CTRL+N**, and then select Proxy Generation. The New wizard dialog displays.
2. From **Create new proxy for**, selected **Android**.
3. Select an existing Android project for the toolkit for which you want to create the proxy.
 **Note:** Depending on the properties of the environment for the selected toolkit, you can generate the code into a project or a folder in the file system.
4. Specify the **Package name**.



5. Choose **Next**. The page for specifying the SAP service you want your application to call displays.



6. Select **Remote location**, to connect to your SAP NetWeaver Gateway server within the network.
7. Click **Catalog** to search for the specific SAP service. The exploration dialog opens for you to search for the service.

When you click **Catalog**, without providing a specific URL, the specified default SAP NetWeaver Gateway is automatically selected.

If you have not saved the password of the default SAP NetWeaver Gateway, a popup displays requesting the password.

Alternatively, select **File system**, and click **Browse** to specify the paths to both the service metadata document for the specific service, and the service document in your file system.

Click **Validate** to verify that the service documents are properly formed XML files for OData compliant services.


Note: If the specified URL of the service metadata comes from the Search Console, the Validate button is not available, as both the connection and the metadata document is verified.

The Validate button is available only if you have manually entered the URL, or selected the metadata file from the file system.

8. Choose **Finish**.

The proxy wizard generates code into the specified Android project.

In addition, it automatically creates class files for the selected SAP service regardless of the specified target environment. The number of files created depends on the number of entities in the selected SAP service.

 Note: Do not locate different proxy contents for the same service into the same project folder, as Eclipse overwrites the files with similar names in a project.

Below is an example of the contents generated for the SAP service, *RMTSAMPLEFLIGHTService*.

The generated proxy consists of the following:

- SRC folder
This folder contains the proxy classes for the selected SAP service in the following packages:
- com.sap.example
This is the package you created for the Android project. It contains the main activity class which is created automatically.

- `com.sap.example.android`
This is the package you created for use in the Android extension. The service class is created.
- `com.sap.example.android.complextypes`
This package contains the Java classes of complex types for the extension.
- `com.sap.example.android.entitytypes`
This package contains the Java classes for entity types.
- `com.sap.example.android.helpers`
This package contains the Java classes of helper methods for the runtime of the Android application, such as connectivity to SAP NetWeaver Gateway.
- **Referenced Libraries**
Contains the libraries (Android OData SDK) of the SAP OData Mobile Client SDK.
- **Res**
This folder contains the service metadata document and the service document XML as well as the file, `service.properties`, that holds the connectivity details for SAP NetWeaver Gateway.

You can call the generated proxy classes from your application to interface with the selected SAP service through the specified SAP NetWeaver Gateway server.

Note that, you have to create your own user interface for rendering your application.



The following is an example of a class with a main method that calls the generated classes and methods for the SAP service, `RMTSAMPLEFLIGHTService`.

- Each line in the example has comments to help you understand the expected process:

```
public class ProxyTestActivity extends Activity implements
ISDMNetListener
{
    SDMLogger logger;
    RMTSAMPLEFLIGHTService service;
    SDMConnectivityHelper connection;
```

- Called when the activity is first created.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    try
    {
        logger = new SDMLogger();
```

- Creates the service, `RMTSAMPLEFLIGHTService`.

```
service = new RMTSAMPLEFLIGHTService(this.getApplicationContext());
```

- Creates the query, `FlightCollection`, for Gateway.

```
ODataQuery flightCollectionQuery = service.getFlightCollectionQuery();
```

- Creates the connection details for Gateway

```
connection = new SDMConnectivityHelper("<username>", "<password>",
this.getApplicationContext());
```

- Executes the request to Gateway

```
connection.executeBasicAsyncRequest(flightCollectionQuery, this);
}
catch (SDMPreferencesException e)
{
    e.printStackTrace();
}
catch (SDMParserException e) {
    e.printStackTrace();
}
catch (MalformedURLException e)
{
    e.printStackTrace();
}
}
```

- Error handling method.

```
@Override
public void onError(ISDMRequest arg0, HttpResponse arg1,
    ISDMRequestStateElement arg2)
{
    logger.i("TAG", "IOException", "onERROR()");
}
```

- Success response method.

```
@Override
public void onSuccess(ISDMRequest aRequest, HttpResponse aResponse)
{
```

- Gets the response from Gateway.

```
    final HttpResponse response = aResponse;
    logger.i("TAG", "Successfully got the response.",
"onSuccess()");final Context context = this.getApplicationContext();
    new Thread()
    {
public void run()
    {
        String respBody = "";
        try {
```

- Getting the entity set as a String

```
        final String entityString =
            EntityUtils.toString(response.getEntity());
List<Flight> flightCollection =
            service.getFlightCollection(entityString);
```

- Gets the first flight in the collection

```
Flight flight = flightCollection.get(0);
```

- Gets the query for the flight's bookings (navigation property query)

```
ODataQuery flightBookings = flight.flightBookingsQuery();
```

- Execute the request for the above query

```
connection.executeBasicAsyncRequest(flightBookings, new
RequestDelegate(context));
```

- Gets the response in RequestDelegate class

```
    } catch (IOException e)
    {
        logger.e("TAG", "IOException", e, "onSuccess()");
    } catch (SDMParseException e) {
        e.printStackTrace();
    }
    }
    }.start();
    }
}
```

Sample of RequestDeligate class:

The navigation property response handler

```
public class RequestDeligate implements ISDMNetListener
{
    SDMLogger logger;
    RMTSAMPLEFLIGHTService service;
    SDMConnectivityHelper connection;
    Context context;
    public RequestDeligate(Context context)
    {
        this.context = context;
    }
    @Override
    public void onError(ISDMRequest arg0, HttpResponse arg1,
        ISDMRequestStateElement arg2)
    {
        logger.e("TAG", "IOException", "onError()");
    }
    @Override
    public void onSuccess(ISDMRequest arg0, HttpResponse arg1)
    {

```

Note: we are not in the UI thread now!

```
        final HttpResponse response = arg1;
        logger.i("TAG", "Successfully got the response.", "onSuccess()");
        new Thread() {
            public void run() {
                String respBody = "";

```

- Response from the navigation property

```
        try {
            final String entityString = EntityUtils.toString(response
                .getEntity());
            service = new RMTSAMPLEFLIGHTService(context);
            Booking flightbooking = Flight.flightbooking(entityString);

```

- Writing the agency number for example to the log

```
        Log.i("TAG", flightbooking.getAGENCYNUM());

```

- Executing another request to Gateway

```
        connection.executeBasicAsyncRequest(service.GetFlightDetailsQuery(
            flightbooking.getcarrid(),
            flightbooking.getconnid()), new FIRequestDeligate(context));

```

- Getting the response in FIRequestDeligate class

```
    } catch (IOException e) {
        logger.e("TAG", "IOException", e, "onSuccess()");
    } catch (SDMParserException e) {
        e.printStackTrace();
    } catch (SDMPreferencesException e) {
        e.printStackTrace();
    }
    }
    }
    }.start();
    }
    }

```

Sample of FIRequestDeligate class:

Function import response handler

```
public class FIRequestDeligate implements ISDMNetListener {
    SDMLogger logger;
    RMTSAMPLEFLIGHTService service;
    SDMConnectivityHelper connection;
    Context context;
    public FIRequestDeligate(Context context) {
        this.context = context;
    }
    @Override
    public void onError(ISDMRequest arg0, HttpResponse arg1,
        ISDMRequestStateElement arg2) {
        logger.i("TAG", "IOException", "onERROR()");
    }
    @Override
    public void onSuccess(ISDMRequest arg0, HttpResponse arg1) {
```

Note: we are not in the UI thread now!

```
        final HttpResponse response = arg1;
        logger.i("TAG", "Successfully got the response.", "onSuccess()");
        new Thread()
        {
            public void run() {
                String respBody = "";
                try {
                    final String entityString = EntityUtils.toString(response.getEntity());
                    service = new RMTSAMPLEFLIGHTService(context);
                    FlightDetails getFlightDetails = service.GetFlightDetails(entityString);
```

- Writing to the log the airport the flight came from

```
        Log.i("TAG", getFlightDetails.getairportFrom());
    } catch (IOException e)
    {
        logger.e("TAG", "IOException", e, "onSuccess()");
    } catch (SDMParseException e)
    {
        e.printStackTrace();
    } catch (SDMPPreferencesException e)
    {
        e.printStackTrace();
    }
    }
    }.start();
    }
}
```

- Create: initialization of the service

```
service = new RMTSAMPLEFLIGHTService(this.getApplicationContext());
```

- Retrieval of the Query object for the bookingCollection

```
ODataQuery bookingCollectionQuery = s.getBookingCollectionQuery();
Date fldate = m_ISO8601Local.parse("2012-06-13T00:00:00");
```

- Initialization of the Booking object

```
Booking booking = new Booking("AA", "0017", fldate, "00001111");
booking.setCUSTOMID("00004617");
booking.setCOUNTER("00000000");
booking.setAGENCYNUM("00000001");
booking.setPASSNAME("Gal Rot");
booking.setPASSFORM("1234567");
booking.setCUSTTYPE("p");
booking.setWUNIT("KG");
booking.setCLASS("Y");
booking.setFORCURAM(new BigDecimal(879.82));
booking.setFORCURKEY("USD");
booking.setLOCCURAM(new BigDecimal(879.82));
booking.setLOCCURKEY("USD");
booking.setPASSBIRTH( m_ISO8601Local.parse("1990-10-10T00:00:00"));
```

- Initialization of the *ConnectivityHelper* object

```
con = new SDMConnectivityHelper("qa", "abc123",this.getApplicationContext());
```

- Execution of the create command against the collection URL using, *bookingCollectionQuery*. The *booking.getStringPayload()* returns the payload of the entry.

```
con.executeAsyncCreateRequest(bookingCollectionQuery,
    booking.getStringPayload(), new CreateDeligate(this.getApplicationContext()));
```

- Update: changing the Passenger's name in the booking object

```
bookingCollectionEntry.setPASSNAME("dany s");
```

- Execution of the update command (against the URL of the booking entry using *bookingCollectionEntry.getEntitysQuery()* – that would return the entry URL)
- The *booking.getStringPayload()* returns the payload of the entry.

```
con.executeAsyncUpdateRequest(bookingCollectionEntry.getEntitysQuery(),
    bookingCollectionEntry.getStringPayload(), new CreateDeligate(context));
```

- Delete: the booking object is the same as the update method. Execution of the delete command (against the URL of the booking entry using *bookingCollectionEntry.getEntitysQuery()* – that would return the entry URL. The *booking.getStringPayload()* returns the payload of the entry.

```
con.executeAsyncDeleteRequest(bookingCollectionEntry.getEntitysQuery(),
    bookingCollectionEntry.getStringPayload(), new CreateDeligate(context));
```

Running your Android Application

You can run the Android application immediately or edit the code in the generated class, and perform various tests for it.

Run and test the generated Android application in the Android-provided simulator in Eclipse.

Runtime Look and Feel of your Starter Application

The runtime of your starter application is based on the default user interface (UI) provided by the design time template for rendering pages.

You can move from a Details page to another page by selecting the title of the page.

In addition, you can activate the following media elements, only if the element had been defined in the service document definition to contain the attribute *sap:semantics*:

TEL:

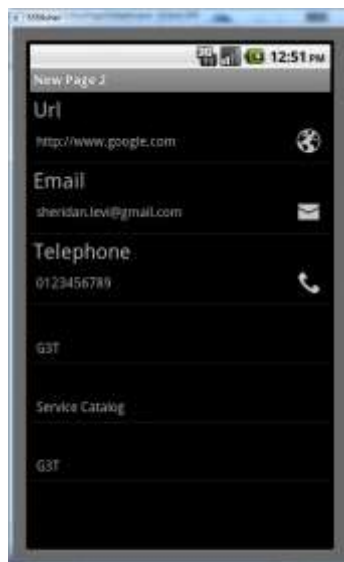
Where the service document used at design time contains a property with the attribute *sap:semantics=tel*, the runtime of the application will contain a telephone icon, and choosing the telephone number activates the call.

EMAIL:

Where the service document used at design time contains a property with the attribute *sap:semantics=email*, the runtime of the application will contain an envelope icon, and choosing the email activates the defined email client. If no email client is defined, you receive the relevant message.

URL

Where the service document used at design time contains a property with the attribute *sap:semantics=URL*, the runtime of the application will contain a circular icon, and choosing the URL activates the Web browser to open the specific URL address.



Secure Android Application

The generated Android application supports basic authentication that uses the browser pop-up dialog for user credentials.

You can use basic authentication over SSL.

Prerequisites

Make sure that you have an SSL server certificate (*Base64 encoded X.509 (.CER)*) from a certificate authority.

For your production environment, verify that, the certificate matches that of the SAP NetWeaver Gateway production landscape.

To use basic authentication over SSL, add the following to the constructor section in the file, *SDMConnectivityHelper.java*:

Read the certificate

```
InputStream is;  
is = resources.openRawResource(com.sup.test.R.raw.sslcer);  
CertificateFactory certFact;  
certFact = CertificateFactory.getInstance("X509");  
X509Certificate cert = (X509Certificate)  
certFact.generateCertificate(is);  
is.close();
```

Add the certificate to the *SDMPreferences*.

```
mParameters.setServerCertificate(cert);
```

Note that the server certificate referenced in the code, *sslcer*, must be located in the folder: *.../res/raw*.

Working with the Sybase Unwired Platform Server

Sybase Unwired Platform (SUP) server is a middleware which you can install and locate between SAP NetWeaver Gateway and your generated Android application.

It is used in a productive landscape where you want to make your generated Android application (both starter application and proxy) available to a large number of people.

Currently, the Android Toolkit provides APIs that enable you to directly connect to the SAP NetWeaver Gateway environment in order to test and evaluate your generated Android application.

Use the information in this section to connect to an SUP server in your SAP NetWeaver Gateway landscape.

Prerequisites

Make sure that you have the following:

- Installed SUP server.
- A user with permissions to work in both the SUP Server and the SAP NetWeaver Gateway.
- Information about the SUP server:
 - The host name and the port number of the SUP server.
 - The Application ID, the name of the registered application.
 - The Farm ID, the default value is 0.
 - The name of the Application Security Configuration.
- You can obtain the above information from the administrator of the SUP server (Sybase Control Center):

The following is an overview of the tasks to perform to enable your Android application to use the SUP server:

1. Generate your Android application using the toolkit
2. Adding the SUP libraries to your Android project.
3. Manually switch the connection to the SUP server.

Adding the SUP Server Libraries to your Android Project

You can enable your generated application to connect to Gateway through the SUP server, using the appropriate SUP server libraries and their dependencies.

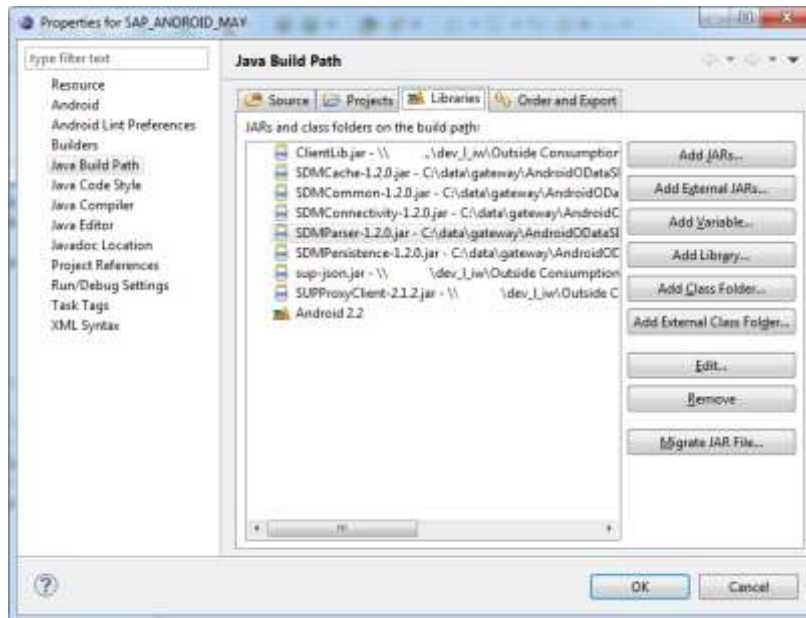
You must add the libraries and their dependencies into the Android project you want to generate.

Find the following files of the SUP server libraries in your file system in the folder, for example, `<SUPUnwiredPlatform_InstallDir>MobileSDK\OData\Android\libraries\`:

- ClientLib.jar
- sup-json.jar
- SUPProxyClient-2.1.2.jar

To add the SUP server libraries:

1. Right click your **Android project**, and choose **Build Path** → **Configure Build Path**.
2. Click the tab, **Libraries**, choose **Add External JARS...**, and select the JAR files in the client libs folder.
3. Select the tab, **Order and Export**, and export the SUP server libraries specified above.
4. Click **OK**.



For more information, go to the [Sybase Infocenter](#).

Manually Switching to the SUP Server

After adding the SUP server libraries and their dependencies to the Android project, you configure the SUP server connectivity.

Prerequisite

You have an Android starter application or the proxy for the Gateway service.

To manually switch to the SUP server, uncomment the following sections in the file, *LoginActivity.java* class:

- The SUP helper class and Properties variable:

```
private SupHelper supHelper;  
private Properties properties;  
try  
{  
    this.isUsingSupServer = true;
```

- Extract the SUP server and application details from the properties file:

```
InputStream rawResource =  
    getApplicationContext().getResources().openRawResource(p.p.R.raw.  
        salesorderservice);  
this.properties = new Properties();  
this.properties.load(rawResource);
```

- The ID of your application:

```
String applicationId = properties.getProperty("applicationId");
```

- The host name of the SUP server:

```
String supHost = properties.getProperty("supHost");
```

- The port number of the SUP server:

```
int supPort = Integer.parseInt(properties.getProperty("supPort"));
```

- Farm id:

```
String farmId = properties.getProperty("farmId");
```

- Create an instance of the SupHelper class:

```
supHelper = new SupHelper (getApplicationContext(), supHost, supPort,
farmId, applicationId);
```

- Check if the user's device is already registered:

```
if (supHelper.getManager().isRegistered())
{
```

- If so, the user name and password are currently empty, and we call "handleActivity":

```
handleActivity("", "");
}
}
catch (SDMPreferencesException e)
{
SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG,
e.getMessage());
}
catch (IOException e)
{
SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG,
e.getMessage());
}
catch (MessagingClientException e)
{
SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG,
e.getMessage());
}
```

- The last "catch" block in "onClick" method:

```
catch (MessagingClientException e)
{
SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG,
e.getMessage());
}
```

- The last exception thrown in "handleActivity" method's signature:

```
private void handleActivity(String userName, String password) throws
SDMPreferencesException, IOException, MessagingClientException
```

The following lines in "*handleActivity*" method:

- **vaultPassword:**
Password of the secure store provided by SDK.
- **securityConfig:**
The security configuration of the application.

```
String vaultPassword = this.properties.getProperty("vaultPassword");
String securityConfig = this.properties.getProperty("securityConfig");
```

- Overwrites the (key, value) pair in the ISDMPreferences interface, with the matching data for use in the SUP server mode:

```
sdmConnectivityHelper.getmPreferences().setStringPreference(ISDMPrefere
nces.SDM_CONNECTIVITY_HANDLER_CLASS_NAME,
"com.sybase.mobile.lib.client.IMOConnectionHandler");
```

- On accessing the application for the first time, the device is registered in the SUP server. Else, perform an unlock action and continue working:

```
supHelper.registerOrUnlock(userName, password, vaultPassword,
securityConfig);
```

- Gets the application's end point from the SUP server and set it as the URL in the service.

```
supHelper.setServiceUrl();
```

From the Android project into which you generated your application, open the automatically generated package called *helpers*, then open the file, *SupHelper.java* and remove all the comments (uncomment).

The *SUPHELPER* class methods:

- Registers the device for the user in the SUP server.

This method will initialize the SUP client with the appropriate data regarding the SUP server. It takes the following:

- username
- password
- securityConfig: the security configuration of the application.

In case of an error, it throws messages from *MessagingClientException*.

```
public void registerUser(String username, String password, String
securityConfig) throws MessagingClientException
{
    if (!liteUserManager.isRegistered())
    {
```

- Registers the specific device once, notice the order of the parameters:

```
liteUserManager.registerUser(username, securityConfig, password);
}
}
```

- Registers the user's device in the SUP server, with vault password.

This method will initialize the SUP client with the appropriate data regarding SUP server. With the option to save the credentials in a secured storage (vault). It takes the following:

- username
- password

- securityConfig: the security configuration of the application.
- vaultPassword: Password of the secure store provided by SDK.

In case of an error, it throws messages from MessagingClientException.

```
public void registerUser(String username, String password, String
securityConfig, String vaultPassword) throws MessagingClientException
{
    if (!liteUserManager.isRegistered())
    {
```

- Registers the specific device once, notice the order of the parameters:

```
liteUserManager.registerUser(username, securityConfig, password,
vaultPassword);
}
}
```

- Unregisters the user's device from the SUP server:

```
public void unregisterSUPUser(LiteUserManager liteUserManager)
{
    if (liteUserManager.isRegistered())
    {
        liteUserManager.deleteUser();
    }
}
```

- On accessing the application for the first time, the device is registered in the SUP server using this method.

Else, unlock the vault and continue working. It takes the following:

- userString
- passwordString
- vaultPassword: Password of the secure store provided by SDK
- securityConfig: the security configuration of the application.

```
public void registerOrUnlock(String userString, String passwordString,
String vaultPassword, String securityConfig) throws
SDMPPreferencesException, MessagingClientException
{
```

- Check if the given user name and password are not empty, meaning they came from the UI:

```
if ((userString.length() > 0) && (passwordString.length() > 0))
{
```

- Registers the user's device:

```
registerUser(userString, passwordString, securityConfig,
vaultPassword);
}
else
{
```

- Extract the user name and password from the vault:

```
manager.unlockVault(vaultPassword);
```

- Start the client:

```
LiteMessagingClient.initInstance(context, appId);
LiteMessagingClient lm = LiteMessagingClient.getInstance();
lm.startClient();
}
}
```

- Gets the application's end point from the SUP server and sets it as the URL in the service:

```
public void setServiceUrl() throws MessagingClientException
{
    LiteAppSettings appSettings = new LiteAppSettings();
```

- Get the URL from the server:

```
String url = appSettings.getApplicationEndPoint();
```

- Set it in the service:

```
SALESORDERService.setUrl(url);
}
```

- Perform Organize imports (CTRL+SHIFT+O) on the entire code before compiling due to compilation errors in the code.

After uncommenting the code, you need to enter information about the SUP server in the file, <service_name>.properties, located in the folder, ...\\res\\raw folder:

- The host name and the port number of the SUP server.
- The Application ID, the name of the registered application.
- The Farm ID, the default value is 0.
- The name of the Application Security Configuration.

```
vaultPassword=<myVaultPassword>
farmId=<myFarmId>
supPort=<myPort>
securityConfig=<mySecurityConfiguration>
supHost=<myHost>
applicationId=<myApplicationId>
```

You can obtain the above information from the administrator of the SUP server (Sybase Control Center).

Android Starter Application on SUP Server

Where your application uses Data Vault or requires the SDK to store the credentials, add the file, *SybaseDataProvider.apk* (this file is part of the SUP Android OData SDK, *AndroidODPSDK*.zip*) to the folder ...\\Data\\APP

Requirement

Make sure that you have added the SUP libraries.

For more information, see *Adding the SUP Server Libraries to your Android Project*.

In addition, do the following:

- Uncomment the content of the entire file, *SupHelper.java*, in the helpers package.
- In the file, *LoginActivity.java* class, remove the comment characters“//” (uncomment) in the following sections:

```
//private SupHelper supHelper;
//private Properties properties;
//try
//{
//this.isUsingSupServer = true;
//// extract the sup server and application details from the properties file
//InputStream rawResource =
getApplicationContext().getResources().openRawResource(p.p.R.raw.salesorders
ervice);
//this.properties = new Properties();
//this.properties.load(rawResource);
//// the id of your application
//String applicationId = properties.getProperty("applicationId");
//// the host name of the sup server
//String supHost = properties.getProperty("supHost");
//// the port number of the sup server
//int supPort = Integer.parseInt(properties.getProperty("supPort"));
//// farm id
//String farmId = properties.getProperty("farmId");
// supHelper = new SupHelper(getApplicationContext(), supHost,
    supPort, farmId, applicationId);
//
//// check if the user's device is already registered
//if (supHelper.getManager().isRegistered())
//{
//// if so, the username and password are currently empty
//handleActivity("", "");
//}
//}
//catch (SDMPreferencesException e)
//{
//SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG, e.getMessage());
//}
```



```

//catch (IOException e)
//{
//SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG, e.getMessage());
//}
//catch (MessagingClientException e)
//{
//SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG, e.getMessage());
//}
//catch (MessagingClientException e)
//{
//SALESORDERApplication.LOGGER.e(SALESORDERApplication.TAG, e.getMessage());
//}private void handleActivity(String userName, String password) throws
SDMPPreferencesException, IOException
/**, MessagingClientException
//the vault password
//String vaultPassword = this.properties.getProperty("vaultPassword");
// the security configuration of the application
//String securityConfig = this.properties.getProperty("securityConfig");
//SDMPPreferences mPreferences = new
//SDMPPreferences(getApplicationContext(), //SALESORDERApplication.LOGGER);
//mPreferences.setStringPreference(ISDMPPreferences.SDM_CONNECTIVITY_HANDLER_
CLASS_NAME,"com.sybase.mobile.lib.client.IMOConnectionHandler");
// supHelper.registerOrUnlock(userName, password, vaultPassword,
securityConfig);
//supHelper.setServiceUrl();
// supHelper.registerOrUnlock(userName, password, vaultPassword,
securityConfig);
//supHelper.setServiceUrl();

```

Runtime of the Generated Android Application

You enable your generated Android starter application or the proxy at runtime to use the SUP server as follows:

- Open the file, <service_name>.properties, in the folder, ../res/raw, and enter the following:

```

vaultPassword=<myVaultPassword>
farmId=<myFarmId>
supPort=<myPort>
securityConfig=<mySecurityConfiguration>
supHost=<myHost>
applicationId=<myApplicationId>

```

- Open the file, LoginActivity.java, and initialize the SupHelper:

```

supHelper = new SupHelper (getApplicationContext(), supHost, supPort,
farmId, applicationId);

```

- Check if the user's device is already registered:

```

if (supHelper.getManager().isRegistered())
{

```

- If so, the user name and password are currently empty, and we call for handleActivity:

```
handleActivity("", "");
}
```

- LoginActivity's handleActivity method.

Enter the following values:

```
String vaultPassword = this.properties.getProperty("vaultPassword");
String securityConfig = this.properties.getProperty("securityConfig");
```

- Overwrites the (key, value) pair in the ISDMPreferences interface, with the matching data for use in the SUP server mode:

```
sdmConnectivityHelper.getmPreferences().setStringPreference(ISDMPreferences.SDM
CONNECTIVITY_HANDLER_CLASS_NAME, "com.sybase.mobile.lib.client.
IMOCConnectionHandler");
```

- On accessing the application for the first time, the device is registered in the SUP server. Else, perform an unlock action and continue working:

```
supHelper.registerOrUnlock(userName, password, vaultPassword,
securityConfig);
```

- Gets the application's end point from the SUP server and sets it as the URL in the service:

```
supHelper.setServiceUrl();
```

- Create an instance of SDMConnectivityHelper and set it in the application:

```
sdmConnectivityHelper = new SDMConnectivityHelper(userName, password,
getApplicationContext());
application.setConnectivityHelper(sdmConnectivityHelper);
```

- Create a query to be executed, for example CurrencyCollection:

```
ODataQuery query = application.getService().get Currency CollectionQuery();
```

- Create an Intent for the next page, for example ListCurrencyActivity:

```
Intent intent = new Intent(getApplicationContext(),
ListCurrencyActivity.class);
```

- Add the data to the Intent:

```
intent.putExtra(SALESORDERApplication.NAVIGATION_URL, query.getUrl());
```

- Start the activity and go to the next page:

```
startActivity(intent);
```

- In the next page's "OnCreate()" method extract the URL of the query from the intent:

```
String url = getIntent().getStringExtra(SALESORDERApplication.NAVIGATION_URL);
```

- Create a new query with the URL:

```
ODataQuery query = new ODataQuery(url);
```

- Execute the request:

```
application.getConnectivityHelper().executeBasicAsyncRequest(query, this);
```

- Get the response in "OnSuccess()" method:

```
public void onSuccess(ISDMRequest aRequest, HttpResponse aResponse)
{
final String data = EntityUtils.toString(response.getEntity());
```

- Parse the data into a list of Currency entries:

```
entries = SALESORDERService.getCurrencyCollection(data);
}
```

- If an error occurs, the response is received in the method, "OnError()":

```
public void onError(ISDMRequest request, HttpResponse response,
ISDMRequestStateElement arg2)
```

```
{  
  //TODO: Perform error handling.  
}  
});
```

Creating Your Own Custom Android Template

You can create a new Android template for use in the framework. The new template is based on the appropriate pattern and environment of the Android toolkit in the framework.

Requirements

Make sure that you have:

- Added the correct plug-in dependency for the Android Toolkit, ***com.sap.iw.gw.oc.eclipse.environment.android***
- Specified the correct environment identifier attribute for the Android Toolkit, ***com.sap.iw.gw.oc.eclipse.environment.android***

For more information, see **Implementing a New Template**.

You can extend the new Android template which receives the environment from the framework.

The following is an example extending the template.

- Proxy template for an Android application setDependencies()

```
@Override
public void setDependencies(Environment environment, Pattern pattern)
{
    // this method is deprecated
}
```

- Android Starter Application template onFinish().

```
@Override
public void onFinish(IProgressMonitor monitor) throws TemplateException
{
    try
    {
        monitor.beginTask(Messages.AndroidList_0, 3);
    }
}
```

- Create a new Android project.

```
this.listPattern = (ListPattern)
this.context.get(TemplateConstants.PATTERN);
this.androidEnvironment = (IAndroidEnvironment)
this.context.get(TemplateConstants.ENVIRONMENT);
androidEnvironment.createNewProject(new SubProgressMonitor(monitor, 1));
monitor.worked(1);
```

- Generate an Android list application into the project
- Generate proxy

```
AndroidProxy androidProxy = new AndroidProxy();
androidProxy.setDependencies(this.environment, this.listPattern);
androidProxy.onFinish(new SubProgressMonitor(monitor, 1));
monitor.worked(1);
```

- Create an Android list application.

```
listPattern.getListModel();
monitor.worked(1);
}
catch (EnvironmentException e)
```

```
{  
    throw new TemplateException(e);  
}  
  
finally  
{  
    monitor.done();  
}  
}
```

Using the HTML5 Toolkit

The SAP NetWeaver Gateway plug-in for Eclipse – HTML5 Toolkit is a collection of environments, patterns and templates, suitable for developing SAP solutions for use in any browser environment that supports HTML5 protocol (SAPUI5 and JQuery) including browsers on mobile devices.

Software Prerequisites

Make sure you have the following:

For JQuery mobile:

- SAP NetWeaver Gateway 2.0 SP3 or higher
Connection settings to an SAP NetWeaver Gateway server. For example, the host name and the port number.
User credentials for logging into the SAP NetWeaver Gateway server to which you want to connect.
- SAP NetWeaver Gateway plug-in for Eclipse Framework installed in your system.
For the SAP NetWeaver Gateway plug-in for Eclipse – HTML5 Toolkit, we recommend to use the Eclipse Web Developer Tools plug-in.

For more information go to:
www.eclipse.org/webtools/

For SAPUI5

- SAP NetWeaver Gateway 2.0 SP5
Connection settings to an SAP NetWeaver Gateway server. For example, the host name and the port number.
User credentials for logging into the SAP NetWeaver Gateway server to which you want to connect.
SAP NetWeaver Gateway plug-in for Eclipse Framework installed in your system.
- SAPUI5 Eclipse plugin (use this link) installed in your system.

Installing the HTML5 Toolkit

To install the HTML5 toolkit:

1. Download the installation package to your local file system.
2. Start Eclipse, and from the main menu choose **Help** → **Install new software**. The Install page displays.
3. Choose **Add**. The Add Repository dialog displays.
4. Choose **Archive**.
5. Browse for the **SAPNetWeaverGatewayPluginForEclipse.zip** file, from the location to which it was downloaded.
6. Choose Open.
7. Open the SAP NetWeaver Gateway node and select the **SAP NetWeaver Gateway plug-in for Eclipse – HTML5 Toolkit** checkbox.
8. Choose **Next**.
9. Choose **Next** again.
10. Select the “I accept the terms of the license agreement” radio button and choose **Finish**.
11. On completion, Eclipse prompts you to restart. Restart Eclipse.

Generating an HTML5 Starter Application

You can create an HTML5 starter application that calls SAP services in SAP NetWeaver Gateway.

Using the project wizard, you generate code for HTML5 into a new web project. The web project can be one of the following:

- JQuery

The starter application targets mobile devices and is based on jQuery Mobile.

For more information about jQuery Mobile, go to: <http://jquerymobile.com/>

- SAPUI5

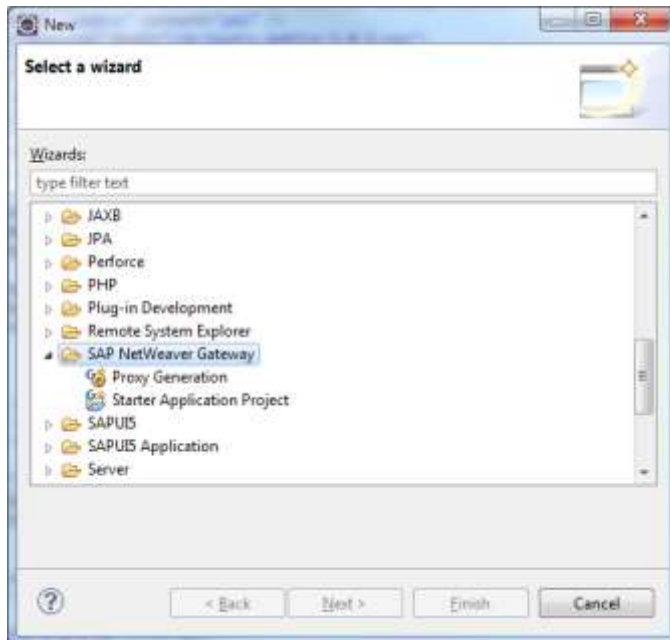
The starter application targets web applications and is based on SAPUI5 protocol.

For more information about HTML5-SAPUI5, go to:

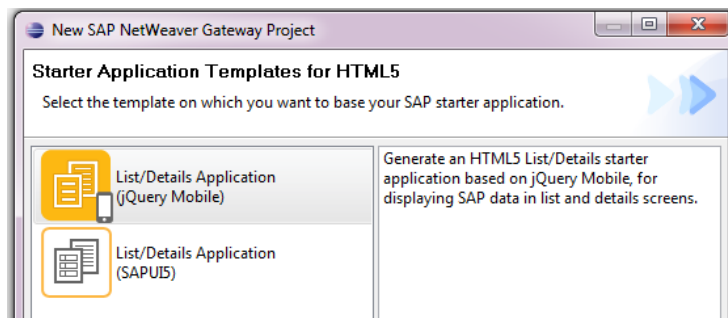
<http://vesapui5.dhcp.wdf.sap.corp:1080/trac/sapui5/wiki>

To create your HTML5 starter application:

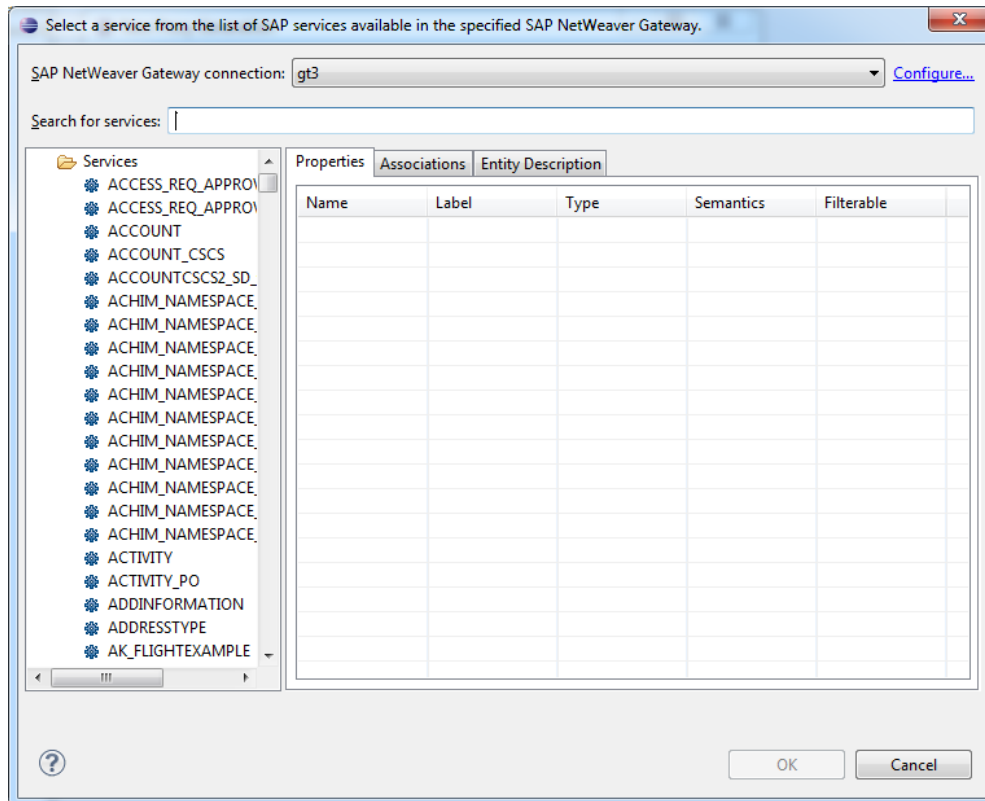
1. From the **File** menu, choose **New** → **Other**. The **New** wizard displays.
2. Select **SAP NetWeaver Gateway** → **Starter Application Project** and choose **Next**.



3. In the Project name field, enter a name for your project. The project and the relevant resources will be created for only your application. For example, SAPCRMCONTACTUI5.
4. From the **Create new project for** drop-down list, select **HTML5**.
5. Choose **Next**. The Starter Application Templates for HTML5 page displays.



6. Select the desired template.
 - List/Details Application for jQuery Mobile
 - List/Details Application for SAPUI5
7. Choose **Next**. The page for specifying the SAP service you want your application to call displays.
8. Choose **Catalog**. The SAP NetWeaver Gateway Services exploration page displays.
9. Specify the SAP NetWeaver Gateway system in which to find the service you want.
10. Search for the desired SAP service.



11. Expand the service group and select the service you want. For example, BANK_DETAILS, and choose **Next**. The List Template-Page Layout page displays.

Note: If you know the Service URL or Service Metadata URL, enter it in the specified field instead of searching in the catalog.


You can also select the File system radio button to browse for the Service Metadata document or the Service Document.

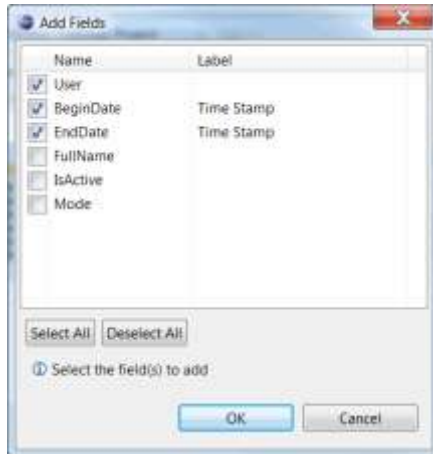
12. Click **Validate** to verify that the service metadata document is a properly formed XML file for OData compliant services.

If the specified URL of the service metadata comes from the Search Console, the Validate button is not available, as both the connection and the metadata document is verified.

The screenshot shows the 'List Template - Page Layout' window. On the left, a 'Pages' list contains several 'New Page 1' entries. The main area on the right has three fields: 'Page Title' (New Page 1), 'Page Type' (List), and 'Entity Set Navigation' (rootCollection). Below these is a table with two columns, 'Name' and 'Label'. At the bottom of the window are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

13. Enter the following:


- JQuery
 - a. In the **Page Title** field, enter the name of the first page.
 - b. From the **Page Type** drop-down list, the List page option is automatically selected.
 - c. From the **Entity Set Navigation** drop-down list, select the desired collection.
 - d. Choose  to see the fields available for the selected entity set. The Add Fields page is displayed.






- e. Select the checkboxes of the desired fields for the page.

Note: We recommend that you not have more than 3-4 fields in a list.

- f. Click OK.

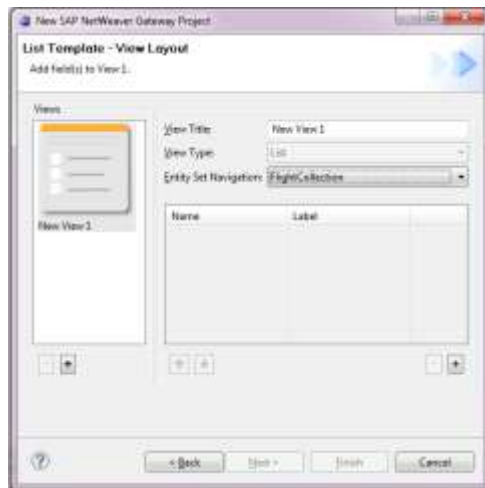
If needed, select a field and click  to delete it.

Use the  and  arrows to change the positioning of the fields in the view. The field positioned first in the list will appear bold in the relevant generated application's list page.


Under the Pages section, choose  to add another page.

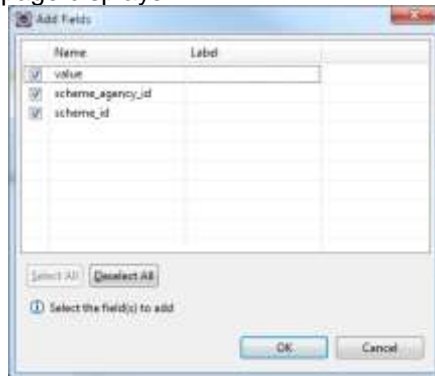
- g. Repeat step 12 to configure the additional pages.
- h. Choose **Finish**. The created project's folder is displayed in the location you previously specified containing code for each page that was created.

- SAPUI5






- a. In the **View Title** field, enter the name of the first view.
- b. From the **View Type** drop-down list, the List view option is automatically selected.
- c. From the **Entity Set Navigation** drop-down list, select the desired collection.


Choose  to see the fields available for the selected entity set. The Add Fields page displays.



- d. Select the checkboxes of the desired fields for the view.
- e. Click **OK**.

If needed, select a field and click  to delete it.

Use the  and  arrows to change the positioning of the fields in the view. The field positioned first in the list will appear bold in the relevant generated application's list view.

Under the Views section, choose  to add another view.

- f. Repeat step **Error! Reference source not found.** to configure the additional views. The new views will generally be allocated 2-per page.
- g. Choose **Finish**. The created project's folder is displayed in the location you previously specified containing code separated into pages, each page contains the views you have specified.

Extending the Generated HTML5 Starter Application

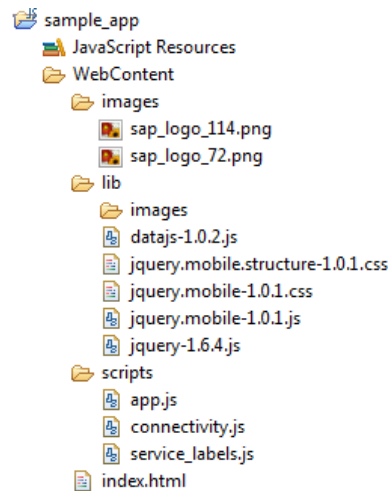
You can modify and extend the generated code for the HTML5 starter application to suit your needs.

The generated project is a Web project, which includes the resources to help in writing HTML and JavaScript. It includes a JavaScript library that can detect syntax errors and provides intellisense.

The main folder is the WebContent folder. This folder contains all the HTML, JavaScript and CSS files required to run the Web application. The number of files created depends on the number of pages you defined in the wizard.

Below are examples of generated projects for the HTML5 extension.

JQuery Mobile



The generated project consists of the following:

- `index.html`

This is the main entry point to the Web application.

The `<head>` element in the file includes references to all the style sheet (CSS) and JavaScript (JS) files needed to run the application.

The `<body>` element renders all the pages of the application. Each page is a `<div>` element with the `data-role` attribute set to `page`.

You can add `<div>` elements to create additional pages or edit existing ones.

- `Lib`

The `lib` folder contains all the external third party libraries and their dependencies.

- `jQuery (jquery-1.6.4.js)`

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript (<http://jquery.com/>).

- `jQuery Mobile (jquery.mobile-1.0.1.js)`

Touch-Optimized Web Framework for Smartphones & Tablets. A unified, HTML5-based user interface system for all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily themeable design (<http://jquerymobile.com/>).

jQuery Mobile also includes the following CSS files:

- `jquery.mobile-1.0.1.css`
- `jquery.mobile.structure-1.0.1.css`
- `Images folder`

The `images` sub folder contains the images required by the jQuery Mobile library.

Caution: Do not remove files from this folder.

- `Datajs-1.0.2.js`

This is an external third party library. It is a new cross-browser JavaScript library that enables data-centric Web applications by leveraging modern protocols, such as, JSON, OData, and

HTML5-enabled browser features. It is designed to be small, fast and easy to use. For more information, visit: <http://datajs.codeplex.com/>.

Updating the libraries:

It is possible to update the above libraries by downloading new versions from their respective sites and replacing the files in the project. Note that you have to update all dependencies (jQuery Mobile is dependent on jQuery) and all references to these libraries in the index.html.

- Scripts

The scripts folder contains all of the JavaScript files generated by the toolkit that are required to run the application.

- app.js

Contains functions used to handle navigation between pages in the index.html file. In addition, it contains global functions for formatting data. This file makes the code in the index.html file cleaner and more concise.

Modify this file if you want to change or add functionality.

- connectivity.js

Contains the functions that handle HTTP calls to the SAP NetWeaver Gateway service by using the dataJS library, and handles request errors that might occur. This file also contains the definition of the service URL and SAP client.

Change this file if the service URL or SAP client changes.

- service_labels.js

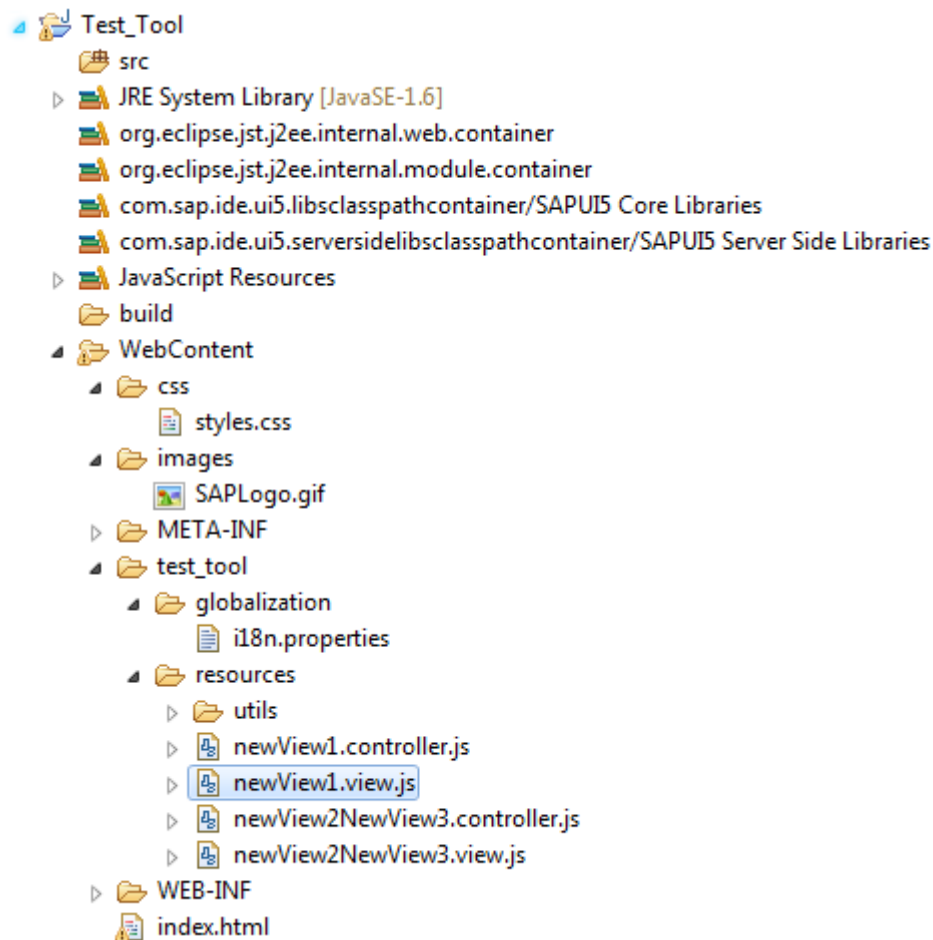
Contains the sap-labels for the properties for all the entities defined in the chosen service metadata. This is used in the file, index.html to display the labels to the user.

- Images folder

The images folder contains SAP logos that are used on iOS devices only when a shortcut is created for the web application.

Replace these icon files with your own company icons to customize the generated app.

SAPUI5



The generated project consists of the following:

- index.html

The file, **index.html**, is the main entry point to the Web application.

The **<head>** element in the file includes references to all the stylesheet (CSS) and JavaScript (JS) files needed to run the application.

The **<body>** element renders all the pages of the application. Each page is a **<div>** element with the data-role attribute set to page.

You can add **<div>** elements to create additional pages or edit existing ones.

Updating the libraries: It is possible to update the above libraries by downloading new versions from their respective sites and replacing the files in the project. Note that you have to update all dependencies and all references to these libraries in the **index.html**.

- Resources

The resources folder contains all of the JavaScript files generated by the toolkit that are required to run the application.

View and controller files.

Contains files for the architecture implementation of the model view controller for each view to be displayed in the generated application.

- connectivity.js

Contains the functions that handle the HTTP calls to the SAP NetWeaver Gateway service by using the dataJS library, and handles requests errors that might occur. This file also contains the definition of the service URL and the SAP client.

Change this file if the service URL or SAP client changes.

- Globalization/i18n.properties

Contains the sap-labels for the properties for all the entities defined in the chosen service metadata. This is used in the file, index.html, to display the labels to the user.

- Images folder

The images folder contains SAP logos that are used on the generated application.

Replace these icon files with your own company icons to customize the generated app.

Styling and Themes

JQuery	SAPUI5
<p>The application uses the jQuery Mobile CSS file for runtime user interface styling.</p> <p>The themes of the header, content, and filter of each application page are defined in the <i>mobileinit</i> event handler of the document object, defined at the <head> section of the <i>index.html</i> file:</p> <pre>\$.mobile.page.prototype.options.headerTheme = "a"; \$.mobile.page.prototype.options.contentTheme = "c"; \$.mobile.listview.prototype.options.filterTheme = "c";</pre> <p>You can modify the selected themes as needed.</p> <p>For more information on the jQuery Mobile themes, see: jquerymobile.com/test/docs/api/themes.html</p>	<p>The application uses the SAPUI5 CSS file for runtime user interface styling.</p> <p>The themes of the header, content, and filter of each application page are defined in the event handler of the document object, defined at the <head> section of the <i>index.html</i> file.</p>

Running your HTML5 Application

jQuery mobile

You can run the jQuery mobile application in any Web browser, or edit the code in the generated file, and perform various tests for it.

The generated jQuery mobile application targets mobile devices. There are several options for deploying a web application to mobile devices:

- Run the application on a dedicated web server and navigate to the server from a web browser on the device.
- Run the application on a dedicated web server and build a native device specific app that wraps a web browser control.

- Run the application locally on the mobile device by building a native device specific app that wraps a web browser control, but also contains all the html, css, and js files required to run the app locally on the device.

If you choose to implement options one of the first 2 options, make sure to configure the web server as a reverse proxy for the SAP NetWeaver Gateway server to prevent the browser from blocking your requests due to **Cross-Origin Resource Sharing** (http://en.wikipedia.org/wiki/Cross-origin_resource_sharing).

Due to security reasons, the browser only allows the JavaScript code to access the server from which the page/script originated. In our case, the HTML page/script is provided by the web server, but the OData feed is provided by the SAP NetWeaver Gateway server.

The URLs used for calling SAP NetWeaver Gateway services from the JavaScript code must use the domain of the reverse proxy (host, port, and protocol). The reverse proxy identifies the SAP NetWeaver Gateway path in the request URL (for example: /sap/opu/odata/...) and sends the request to the SAP NetWeaver Gateway server. This server then sends the response directly to the client.

For example, let's assume that the web server URL is:

http://webserver1:1234/myjqmobileapp, and the SAP NetWeaver Gateway service URL is: ***http://gatewayserver:50028/sap/opu/odata/iwfnd/mygwservice***

Open the **connectivity.js** file and change the serviceUrl variable to point to the web server instead of the SAP NetWeaver Gateway server.

Replace this:

```
var serviceUrl =
"http://gatewayserver:50028/sap/opu/odata/iwfnd/mygwservice/";
```

With this:

```
var serviceUrl =
"http://webserver1:1234/sap/opu/odata/iwfnd/mygwservice/";
```

For the second and third options, there are 3rd party solutions which help produce the native wrapper application such as:

- PhoneGap (<http://phonegap.com/>)
- Titanium (<http://www.appcelerator.com/>)
- Corona (<http://www.anscamobile.com/>)

SAPUI5

You can run the SAPUI5 application in any Web browser, or edit the code in the generated file, and perform various tests for it.

Before you deploy the application, you must edit the *index.html* file as follows.

1. Open the the *index.html* file located in the WebContent directory.
2. Search for the following line in the Script tag:
`src="./sapui5-1.4/resources/sap-ui-core.js"`
3. Replace the text in quotation marks by a link to the location where the files for SAPUI5 version 1.4 are located.

The generated SAPUI5 application targets web computers. There are 2 options for deploying the application:

- Run the app on a dedicated web server, and navigate to the server from a web browser.
- Run the app locally, css and js files required to run the app should be handled locally on the device.

If you choose to implement the second option, make sure to configure the web server as a reverse proxy for the SAP NetWeaver Gateway server to prevent the browser from blocking your requests due to **Cross-Origin Resource Sharing** (http://en.wikipedia.org/wiki/Cross-origin_resource_sharing).

Due to security reasons, the browser only allows the JavaScript code to access the server from which the page/script originated. In our case, the HTML-SAPUI5 page/script is provided by the web server, but the OData feed is provided by the SAP NetWeaver Gateway server.

The URLs used for calling SAP NetWeaver Gateway services from the JavaScript code must use the domain of the reverse proxy (host, port, and protocol). The reverse proxy identifies the Gateway path in the request URL (for example: /sap/opu/odata/...) and sends the request to the SAP NetWeaver Gateway server. This server then sends the response directly to the client.

For example, let's assume that the web server URL is:

http://webserver1:1234/myjqmobileapp, and the SAP NetWeaver Gateway service URL is:
http://gatewayserver:50028/sap/opu/odata/iwfnd/mygwservice

Open the connectivity.js file and change the serviceUrl variable to point to the web server instead of the SAP NetWeaver Gateway server.

Replace this:

```
var serviceUrl =  
"http://gatewayserver:50028/sap/opu/odata/iwfnd/mygwservice/";
```

With this:

```
var serviceUrl =  
"http://webserver1:1234/sap/opu/odata/iwfnd/mygwservice/";
```

Security in your HTML5 Application

The generated application supports basic authentication that uses the browser pop-up dialog for user credentials.

Extensibility

The SAP NetWeaver Gateway plug-in for Eclipse Framework provides extension points for creating your own custom extensions as templates, and environments which are added to the tools and wizards of the framework.

Applications for SAP solutions meant to run in different runtime environments can have patterns that you reuse many times.

The SAP NetWeaver Gateway plug-in for Eclipse allows application developers aiming to provide SAP solutions for multiple environments to automatically generate code for their specific target environment, and to develop their applications faster.

The generated source code can be customized according to the specific needs and purposes of the different business solutions.

Overview of Extensibility

SAP NetWeaver Gateway plug-in for Eclipse provides the following main ways of extensibility:

- Implementing new templates

You can create a new template as an extension of the template of the framework. This is added to the specific toolkit.

For more information, see *Implementing a New Template*

- Implementing new environments

You can create a new environment as an extension of the environment of the framework. This is added to the specific toolkit.

For more information, see *Implementing a New Environment*

Implementing a New Template

A template describes the set of coding patterns that can be reused to generate code for applications that run in a specific target environment.

The framework provides its own extension point from which you derive new extension templates. The outcome of each new extension template is the generation of a template class with code for a fully-functional template for a starter application.

Templates are customizable, as you can modify and customize the code generated for the template.

You create a new template by extending the specific extension point, *com.sap.example.eclipse.framework.template*, supplied for templates in the framework.

The following is an overview of the sequence of processes for creating a new template:

1. Adding the plug-in for the framework as the required dependency plug-in.
2. Create a new template class based on the required dependency plug-in.

Prerequisites

Make sure that you have the following:

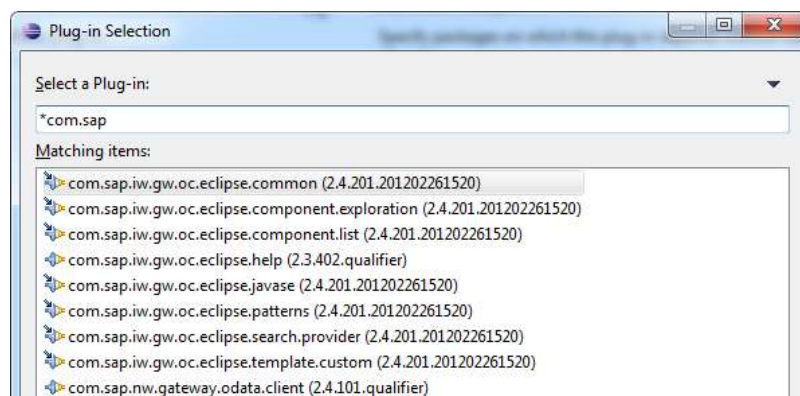
- A new plug-in project in Java.
- The framework plug-in on which your template is dependent.
- The file, *Manifest.mf*, or *plug-in.xml*, is available in your new project.

Adding the Required Dependency Plug-in

Before you create a new template and environment, you must specify the plug-in for the framework, *com.sap.iw.gw.oc.eclipse.framework*, as the plug-in that will contribute code to your extensibility project. The plug-in is required in the classpath of the project in order to compile.

To define the required plug-in dependency:

1. Optionally create a new Plug-in Project in Java if you have not done so.
2. Double click the file, **Manifest.mf**, in your new plug-in project. The Manifest Editor opens.
3. Open the Dependency tab, and click **Add** under Required Plug-in. The Plug-in Selection dialog opens.




4. In Select a Plug-in, enter the term, ***com.sap***, to search for all plug-ins containing the search term.
5. Select the plug-in for the framework, *com.sap.iw.gw.oc.eclipse.framework*, from the list and choose **OK**.

Creating the New Template


After adding the dependency plug-in, you can create a new template that will be dependent on the framework plug-in.

To create the new template class:

1. Optionally create a new Plug-in Project in Java if you have not done so.
2. Double click the file, **Manifest.mf**, under META-INF in your new plug-in project.
3. Open the Extensions tab, and click **Add** under **All Extensions**. The Extension Points Selection dialog displays a list of all the extensions points.
4. In Extension Points filter, enter the term, **template*, to search for all extension points containing the search term.
5. Select the extension point, *com.sap.iw.gw.oc.eclipse.framework.template*, from the list and choose **Finish**. A new extension element with the same name as the extension point is created.
6. Right click the new extension element, select **New**, and then specify the type of template (exists as child elements) you want.


 Recommendation: We recommend that you save your changes after modifying the value of each attribute.

The following template types are available:

- Proxy:
If you choose the Proxy child element, enter the required values for the attributes with asterisks:
 - id: A unique ID for the template.
 - Template: A fully qualified name of a class that will extend, *com.sap.iw.gw.oc.eclipse.framework.template*, interface.
 - environmentId: The ID of an existing environment to be referenced by the new template.
 - patternId: Optionally provide a pattern ID.
- Starter Application:
 -  Recommendation: We recommend that you save your changes after modifying the value of each attribute.

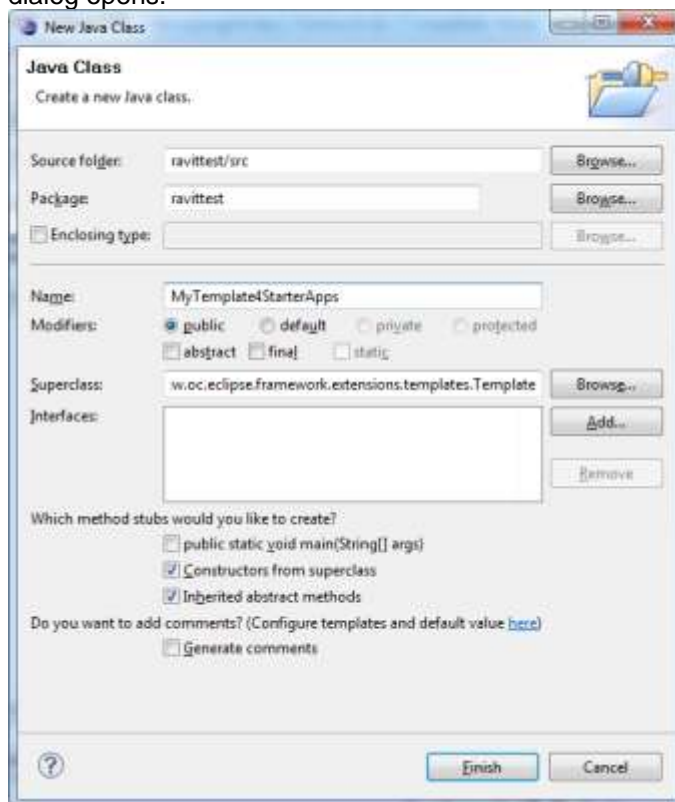
If you choose the Starter Application child element, enter the required values for the attributes with asterisks:

- id: A unique name that will be used to identify this template.
- Template: A fully qualified name of a class that will extend, *com.sap.example.eclipse.framework.template*, interface.
- environmentId: The identifier of an existing environment to be referenced by the new template.

 Note: The environment determines access to the new template, as the template is available for selection in a specific environment.

- displayName: A translatable name that will be used in for the template in the wizard.
- description: Descriptive text that displays information about the new template in the wizard.
- icon: The path of a graphic file that will visually represent the template in the wizard.
- patternId: Optionally provide a pattern ID.

- Click the attribute, **Template**, under **Extension elements** details. The New Java Class dialog opens.



- Enter a name for your template class.
The Superclass points to the class within the framework that must be implemented. The selected modifier is Public, and the selected method stubs are constructors from the superclass and inherited abstract class.
- Click **Finish**, and then click **Save** in the Eclipse main menu.
Code is generated for the new template in the specified template class.

Generating a Starter Application from a Your New Template

You can create a fully functional starter application based on your new template for a specified target environment, as the new template is added to the Starter Application wizard of the framework.

- To run and test your new template, select **Run** from the main menu and choose **Run**.
- Open the Starter Application Wizard of the framework, specify the target environment and provide the required entries for it, and then select your new template to generate a starter application.

Extending the New Template

The template receives an environment and a pattern from the framework, and needs to save them as private members. Some templates do not have a pattern and so they can be null.

You can cast these elements to their specific types in this method or later in a different method, such as, *onFinish()*, for example.

Example Code Snippets

From Basic Starter Application template:

```
@Override
public void setDependencies(Environment environment, Pattern pattern)
{
    // this method is deprecated
}
```

- Returns the template's wizard pages. Can return null.

```
public List<IWizardPage> getWizardPages();
```

- This method is responsible for the logic when you choose the Finish button. You can use the abilities of the environment and pattern models.

```
public abstract void onFinish() throws TemplateException;
```

- From the Java Basic starter application template:

```
@Override
public void onFinish(IProgressMonitor monitor) throws TemplateException
{
    try
    {
        monitor.beginTask("generate Java Basic template", 6);
```

- Create new java project with client library jar

```
this.javaSeEnvironment =
this.context.get(TemplateConstants.ENVIRONMENT);

this.javaSeEnvironment.createNewProject(new SubProgressMonitor(monitor,
1));

monitor.worked(1);

Bundle bundle = Platform.getBundle(Activator.PLUGIN_ID);

Path path = new Path("/templateDocs/custom/CustomDocument.vm"); //$NON-NLS-1$

EnumMap<JavaSeEnvConstants, Object> model =
    this.javaSeEnvironment.getModel();

String fileOutputPath = (String)
model.get(JavaSeEnvConstants.PACKAGE_PATH);

String mainClassJavaFile = (String)
model.get(JavaSeEnvConstants.MAIN_CLASS_NAME);

String javaPackage = (String)
model.get(JavaSeEnvConstants.PACKAGE_NAME);
```

- Building an input model

```
Map<String, Object> inputMap = new HashMap<String, Object>();

inputMap.put("package", javaPackage); //$NON-NLS-1$
```



```
inputMap.put("MAIN_CLASS", mainClassJavaFile); //$NON-NLS-1$
monitor.worked(1);
```

- **Generating a basic Java file**

```
IFileGenerator generator = new FileGenerator();
String filePath = generator.generate(bundle, path, fileOutputPath +
mainClassJavaFile + ".java", inputMap); //$NON-NLS-1$
monitor.worked(1);
String projectName = (String)
model.get(JavaSEEnvConstants.PROJECT_NAME);
```

- **Refresh the generated project**

```
ProjectUtils.refreshProject(projectName);
monitor.worked(1);
```

- **Organize imports**

```
List<String> generatedFilesPathes = new ArrayList<String>();
generatedFilesPathes.add(filePath);
ProjectUtils.organizeJavaImportsAndSave(projectName,
generatedFilesPathes);
monitor.worked(1);
```

- **Opens the generated file**

```
ProjectUtils.openFile(projectName, filePath);
monitor.worked(1);
monitor.done();
}
catch (FileGeneratorException e)
{
throw new TemplateException(e);
}
catch (EnvironmentException e)
{
throw new TemplateException(e);
}
catch (CoreException e)
{
throw new TemplateException(e);
}
}
```

Modifying a Template of a Toolkit

You can modify the templates provided in the toolkit of the framework plug-in and save them as new templates for use in the wizard of the framework.

The changes you make are visible at the runtime of the wizards, and the generated application on which you base the modified template.

The following is an overview of the sequence of tasks for modifying a template:

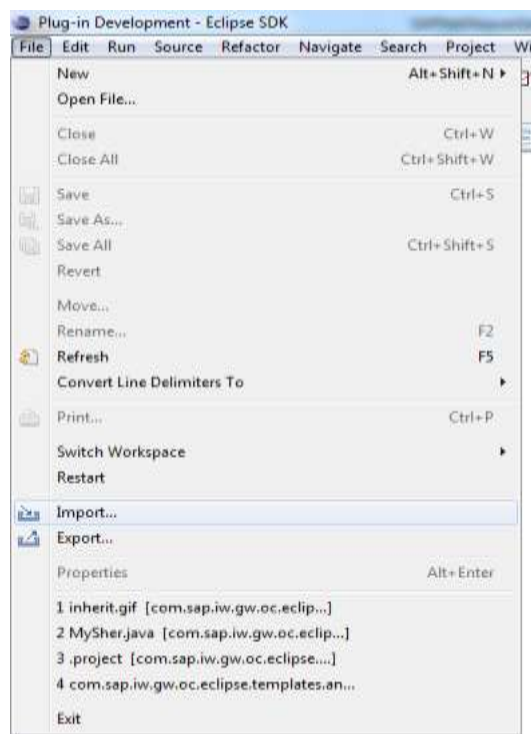
1. Import the template component you want to modify.
2. Rename the project for imported template.
3. Make the changes you want and run the template.

Importing the Template Component of the Toolkit

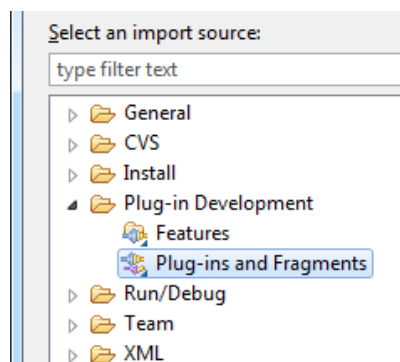
You must import the template component for the specific toolkit you want to modify as a new plug-in development project.

To import the template component:

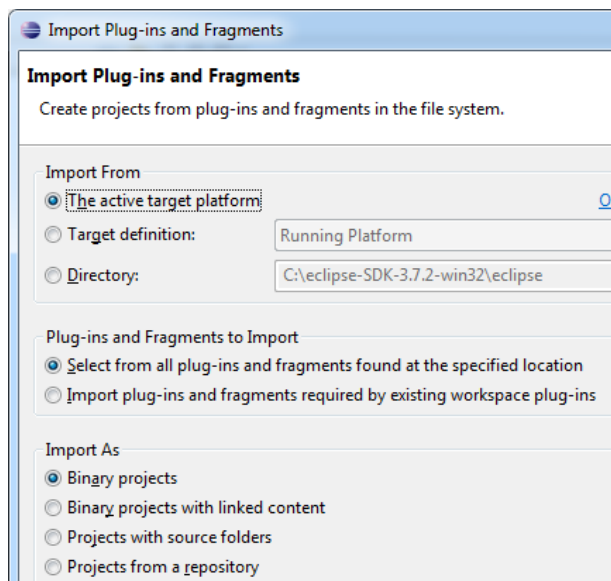
1. From the Eclipse menu, choose **File** → **Import**.



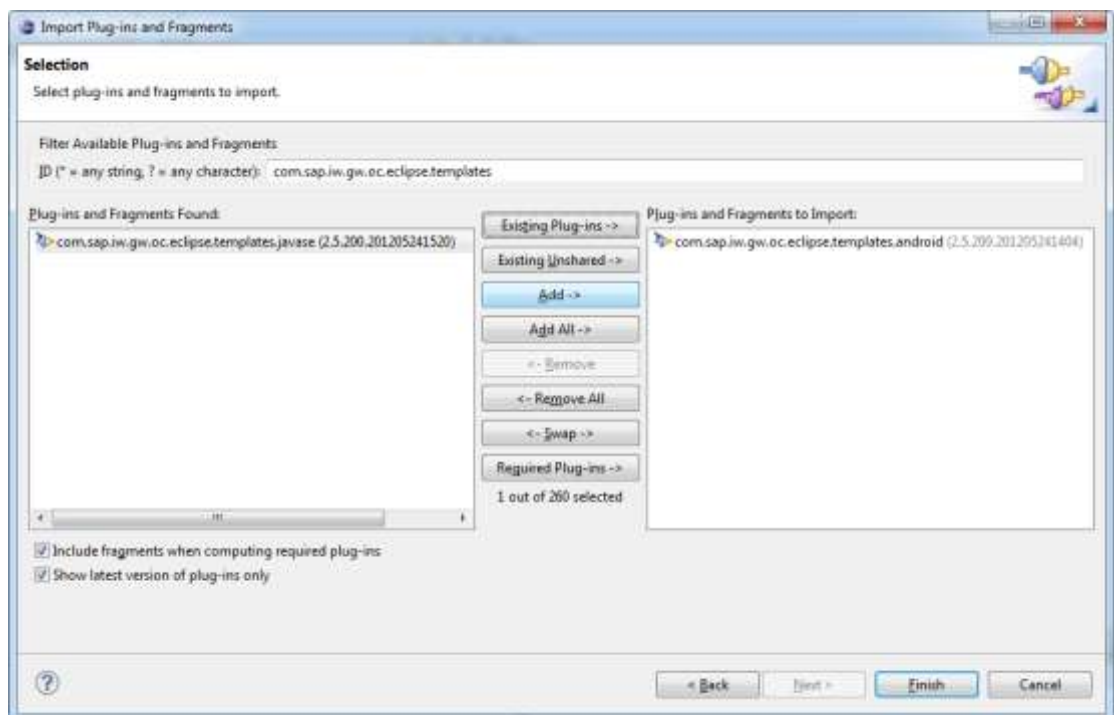
2. Expand **Plug-in and Development** in the Import Source, and choose **Plug-ins and Fragments**.



3. Click **Next**, leave the default values and click **Next**.



4. In ID, enter the search item for the template component of the toolkit you want to modify. For example, *com.sap.iw.gw.oc.eclipse.templates*. and click **Add**, and then click **Finish**.



Eclipse creates a new plug-in project with the name of the selected template component.

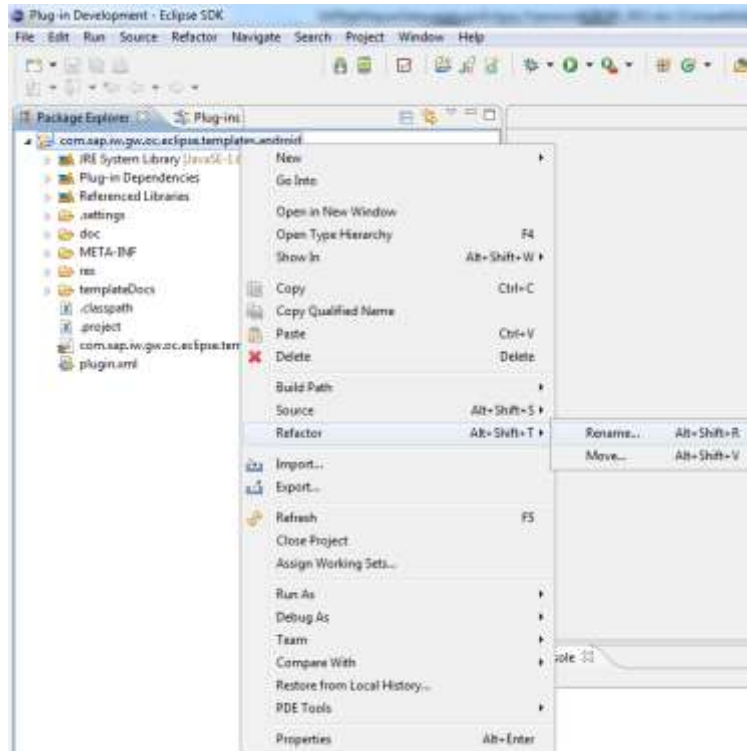


Renaming the Project for the Template Component

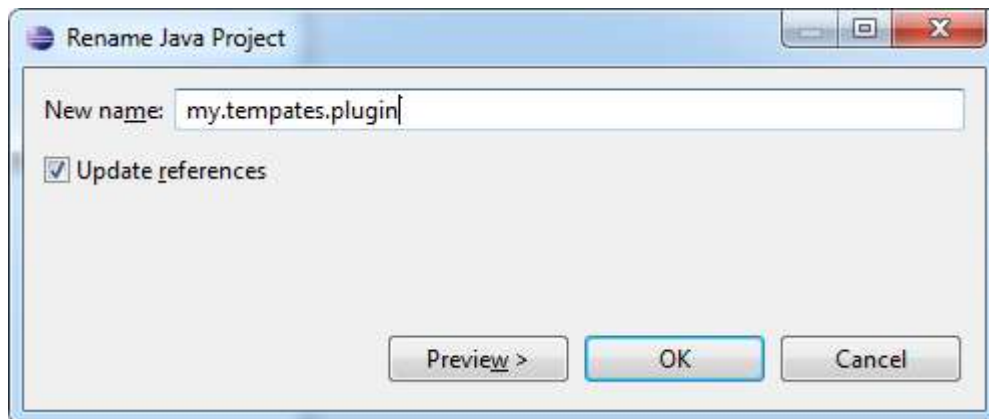
You must rename the project created for the template component.

To rename the project:

1. Right click the project for the new template, and select **Refactor** → **Rename**.



2. Change the name of the project, select **Update references**, and then click **OK**.



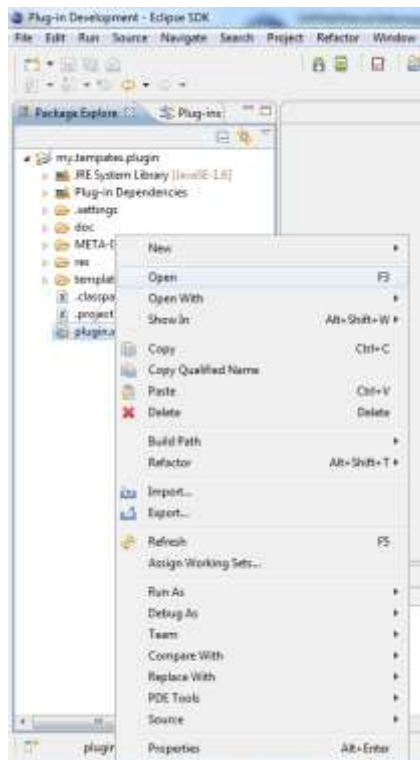
Editing and Running the Template

You can modify the runtime of the template to be displayed in the wizard to show your custom specific logo, template name, description, and many more

In addition, you can change aspects of the template for rendering of generated applications based on the modified template.

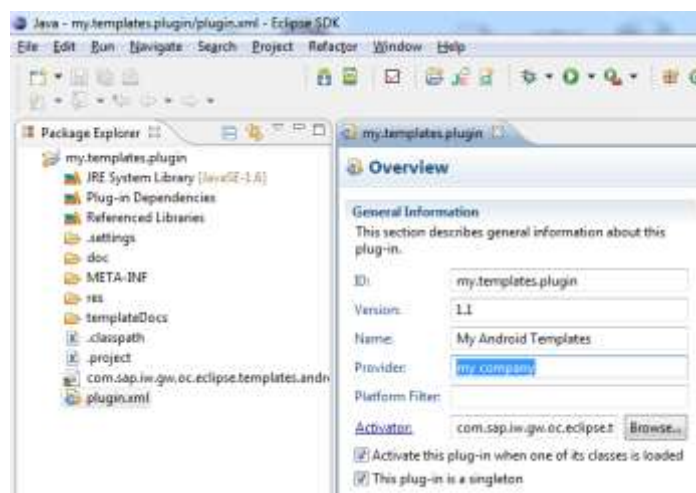
To edit the template for the runtime of the wizard:

1. Expand the project for the template component in the Package Explorer, and double click the file, **Plugin.xml**.



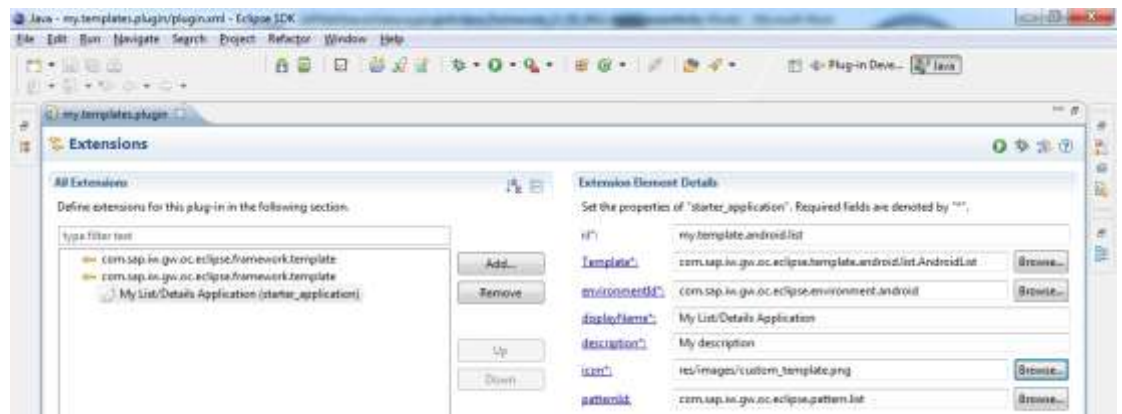
2. We recommend that you change the following:

- Version: Specify your own version number.
- Provider: Specify your organization or company name.
- Name: Specify a name for the template



3. Choose the **Extensions** tab, and specify the following:

- ID: Change the identifier for the template, so that it can be presented in the list of templates of the wizard. The specified ID is already assigned to another template.
- Template: Do not modify the template value.
- Environmentid: Do not modify the environment identifier value.
- displayName: Optionally, change the name of the template. The name you specify is presented as the name of the template in the wizard.
- icon: Optionally, change the icon for the template. The specified image is presented for the template in the wizard.
- Description: Optionally, change the description for the template. The descriptive text is presented for the template in the wizard.
- Patternid: Do not modify the pattern identifier.



4. Click **Save** to save your changes.

Changing the Look and Feel of Applications Generated from a Modified Template

You can make several changes to the look and feel of the runtime of the applications that you generate based on your modified template.

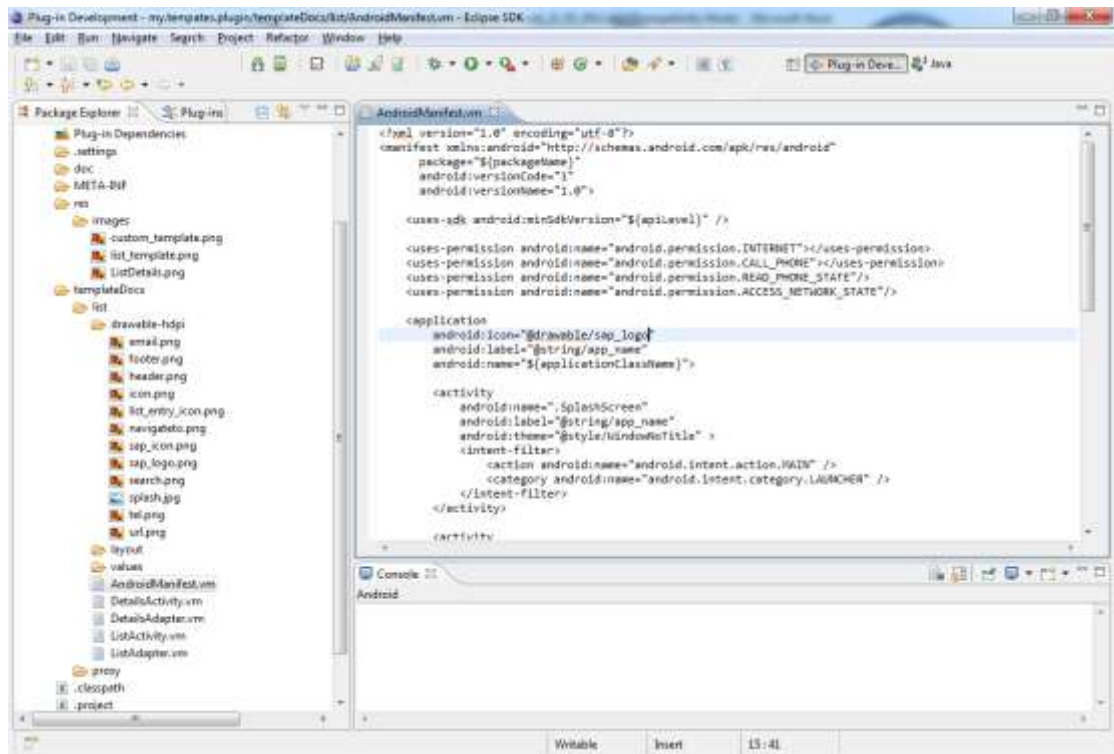
In this section, you will find examples of how to modify the following aspects of the runtime of the generated applications based on the template:

- Display your custom logo for generated applications.
- Change the background color of the layouts for rendered applications.
- Increase the font size for the label for search.
- Change the font color for the text displayed

Example: Display your custom logo for generated Android applications

Make sure that, the image you want to use as a logo for the runtime of the generated application is available in the folder, *templateDocs/drawable_hdpi*

1. From the project, open **templateDoc** → **list**, and open the file, **AndroidManifest.xml**.



2. Go to the section with the following lines:

```
<application
    android:icon="@drawable/icon"
```

3. Replace the name, **icon**, with the name of your logo. For example:

```
<application
    android:icon="@drawable/sap_logo"
```

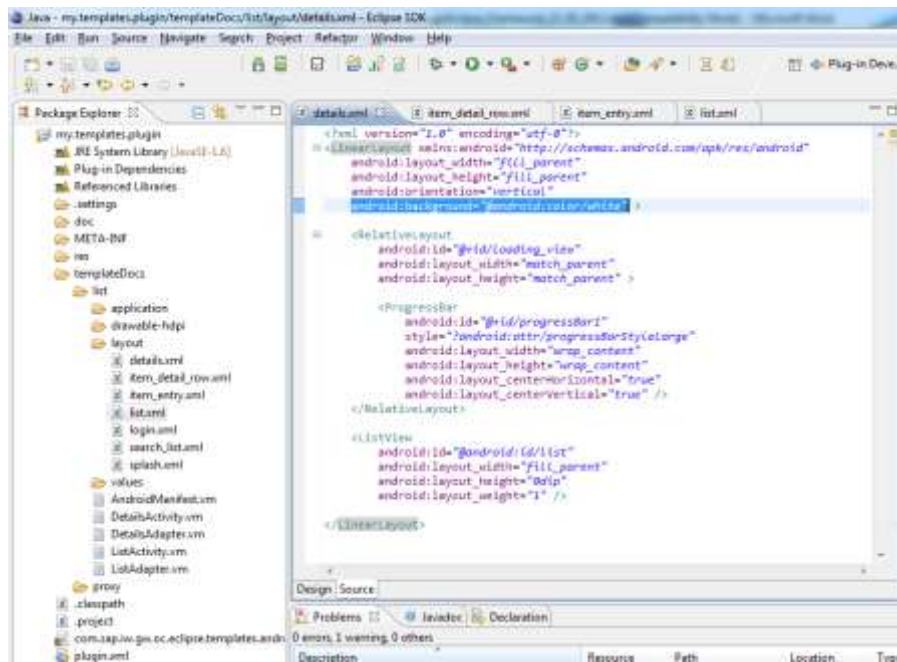
4. Click **Save**.

Example: Change the background color for Android applications

1. From the project, open **templateDoc** → **list** → **layout**.
2. Open the following files:
 - o details.xml:

Go to the section with the following lines and add to the end of the section (within the angular brackets), the line, **android:background="@android:color/white"**:

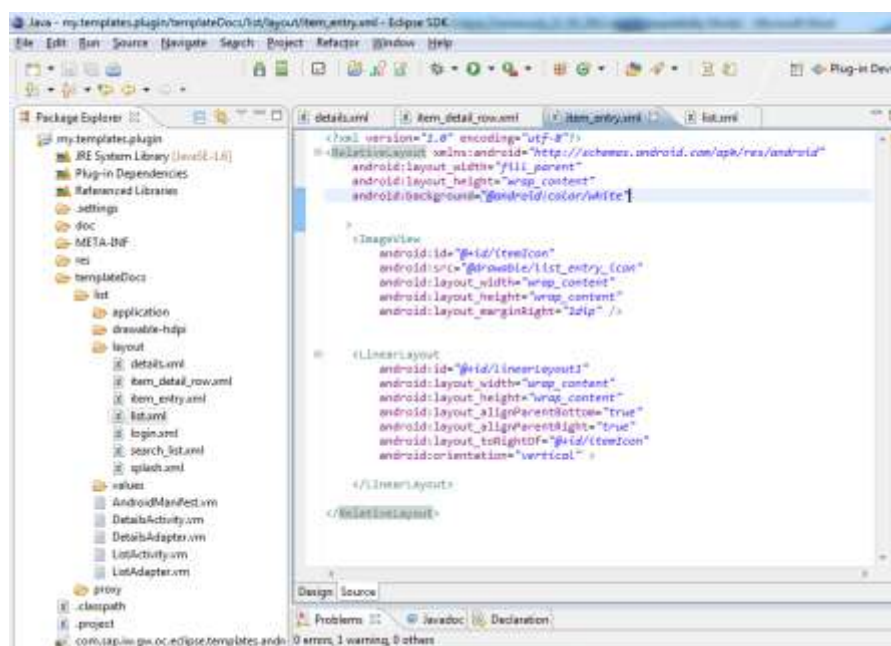
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ....
    android:background="@android:color/white">
```

- item_entry.xml:

Go to the section with the following lines and add to the end of the section (within the angular brackets), the line, **android:background="@android:color/white**.

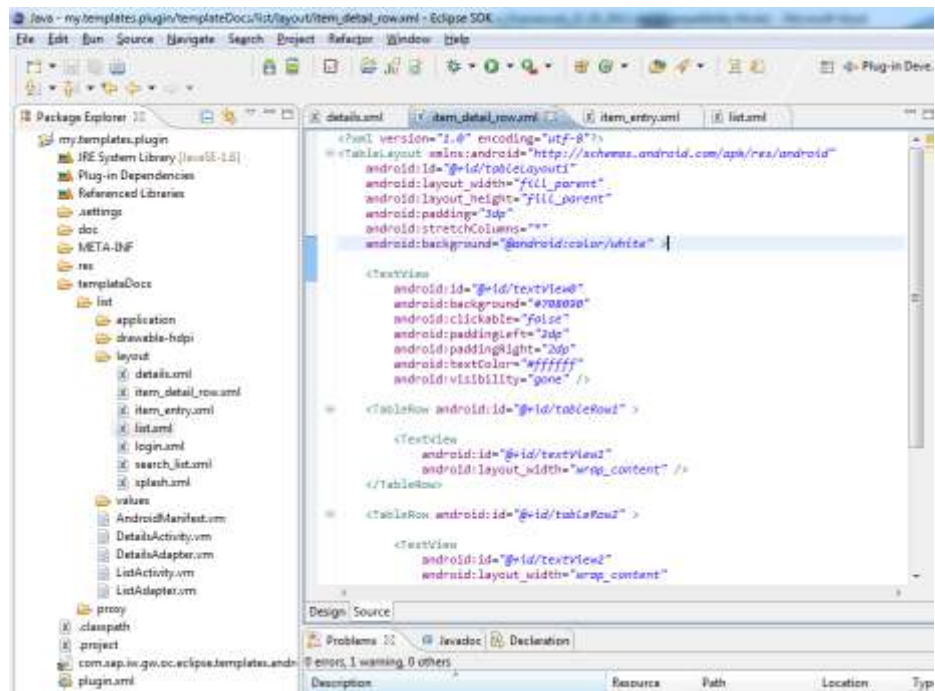
```
<RelativeLayout xmlns:android=http://schemas.android.com/apk/res/android
...
    android:background="@android:color/white" >
```



- Item_detail_row.xml

Go to the section with the following lines and add to the end of the section (within the angular brackets), the following line, **android:background="@android:color/white**.

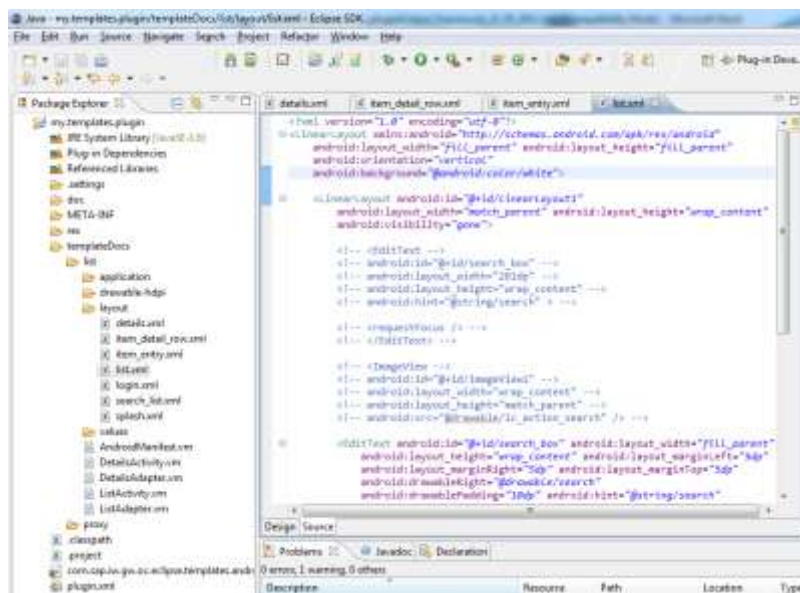
```
<TableLayout xmlns:android=http://schemas.android.com/apk/res/android
...
    android:background="@android:color/white">
```

- List.xml

Go to the section with the following lines and add to the end of the section (within the angular brackets), the line, **android:background="@android:color/white**.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ....
    android:background="@android:color/white">
```

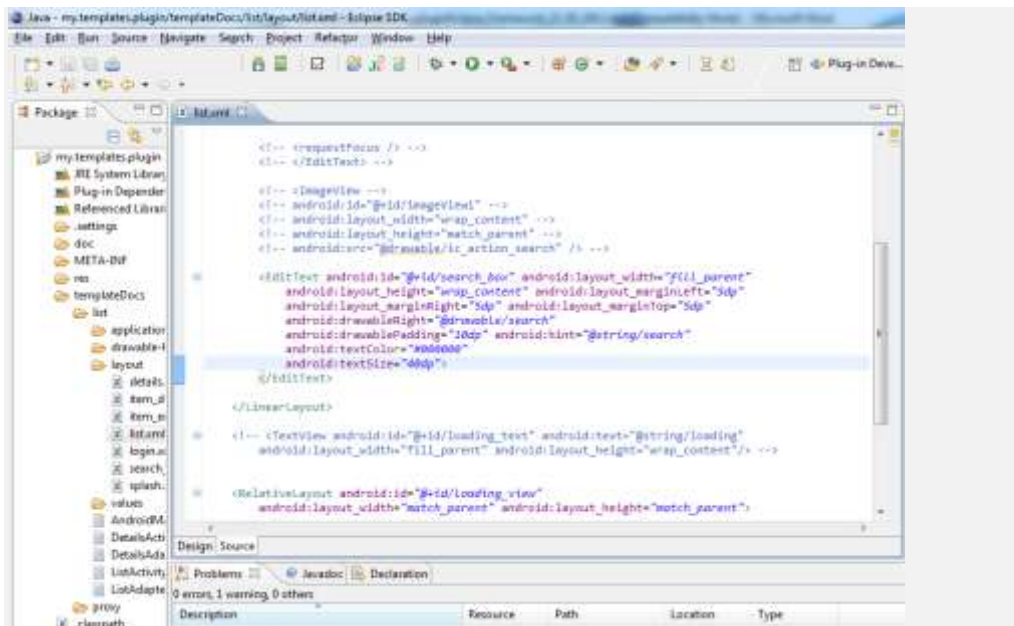


3. Click **Save**.

Example: Increase the font size for the label, search

1. From the project, open **templateDoc** → **list** → **layout**.
2. Open the file, **list.xml**, and go to the section with the following lines and add to the end of the section (within the angular brackets), the line, **android:textSize="40dp"**.

```
<EditText android:id="@+id/search_box" android:layout width="fill parent"
    android:layout height="wrap content" android:layout marginLeft="5dp"
    android:layout marginRight="5dp" android:layout marginTop="5dp"
    android:drawableRight="@drawable/search"
    ...
    android:textSize="40dp">
</EditText>
```



3. Click **Save**.

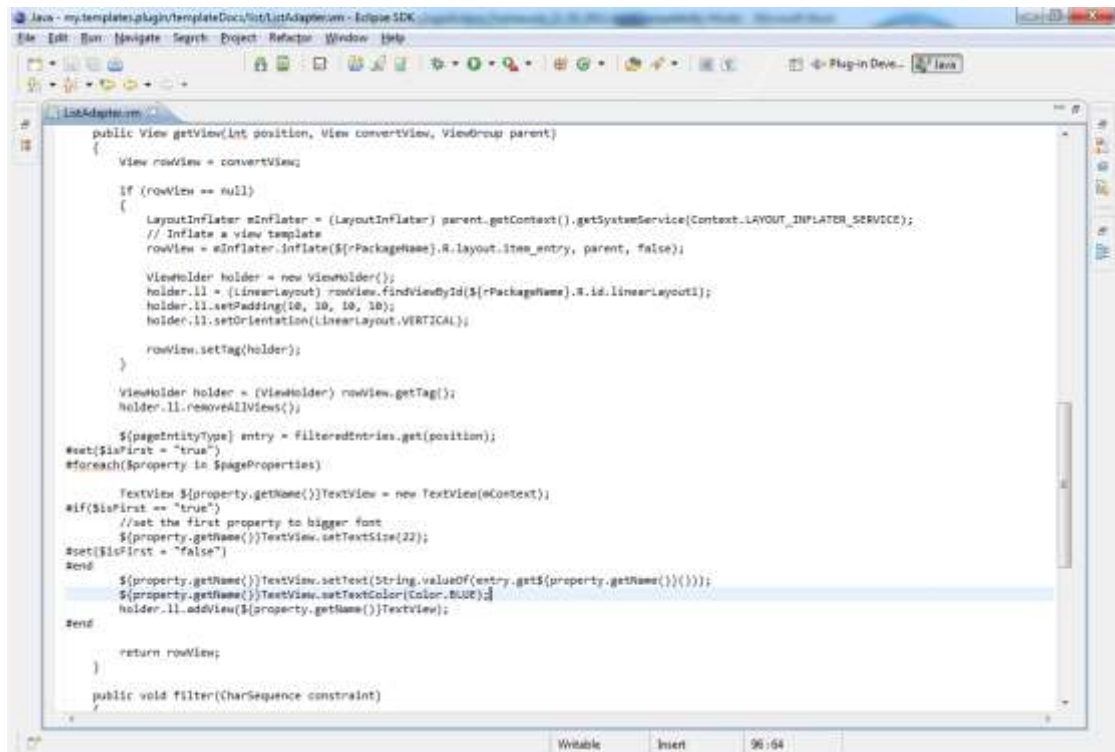
Example: Change the font color for the text displayed for an Android application

1. From the project, open **templateDoc** → **list**.
2. Open the file, **listadapter.vm**, and go to the section with the following lines:
\${property.getName()}TextView.setText(String.valueOf(entry.get\${property.getName()}()
));

Insert the line:

\${property.getName()}TextView.setTextColor(Color.BLUE); after the above line. For example:

```
${property.getName()}TextView.setText(String.valueOf(entry.get${property.getName()}()  
));${property.getName()}TextView.setTextColor(Color.BLUE);  
holder.ll.addView(${property.getName()}TextView);
```



Open the file, **detailsadapter.vm**, and go to the section with the following lines:

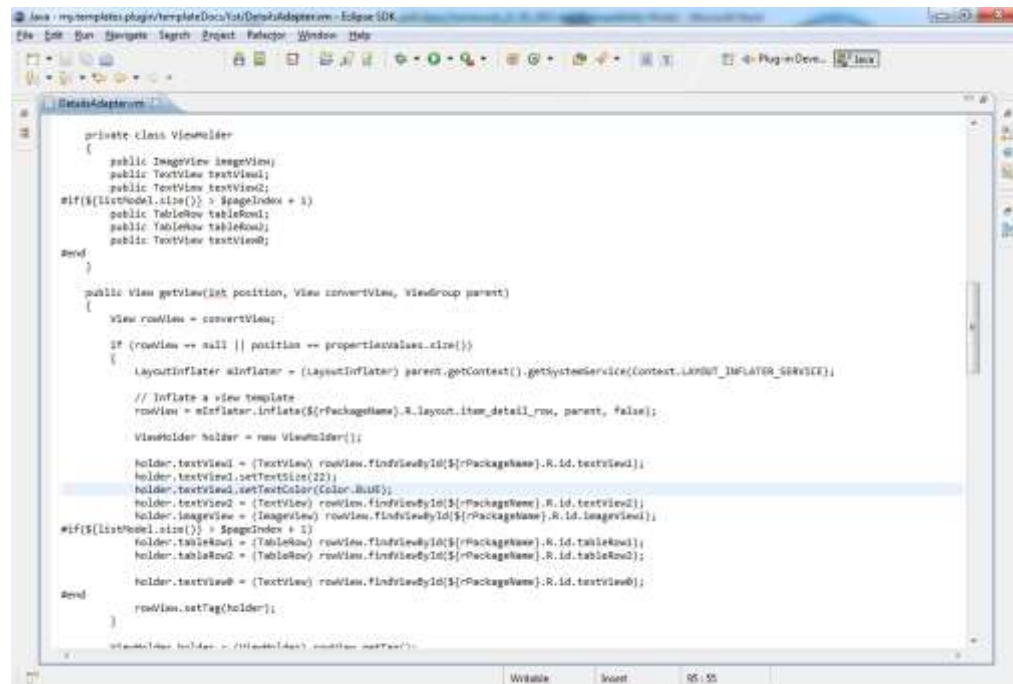
```
holder.textView1 = (TextView) rowView.findViewById(${rPackageName}.R.id.textView1);
```

3. Insert the line, **holder.textView1.setTextColor(Color.BLUE)**; after the above line. For example:

```

holder.textView1 = (TextView) rowView.findViewById(${rPackageName}.R.id.textView1);
holder.textView1.setTextSize(22);
holder.textView1.setTextColor(Color.BLUE);

```

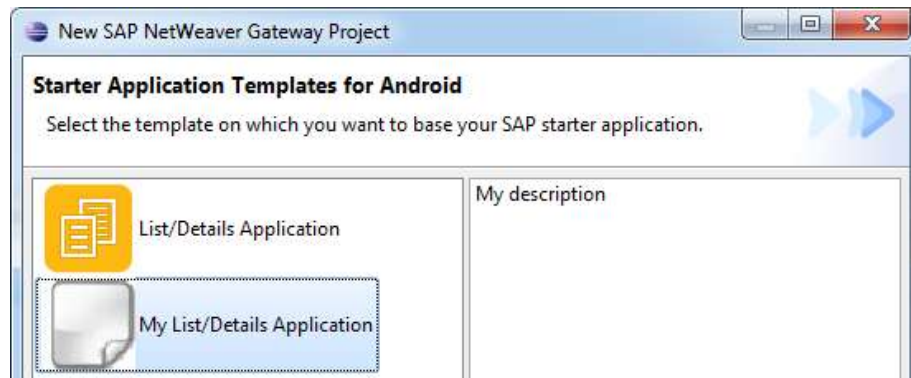


4. Click **Save**.

You can view the changes you made to the template by running it.

To do so:

1. Right click the project and choose **New** → **Run As** → **Eclipse Application** to start the Starter Application wizard.
2. Select the toolkit for which you modified the template to see the template presented in the list of templates, and click **Next**.




For more information on how to generate a starter application based on a modified template for a toolkit, go to the specific section about the toolkit.

Example: Editing Your Custom Template

You can change the text, the labels, and the icons provided in the default template supplied with your toolkit in the list pattern model of the plug-in.

For example, the Android Toolkit comes a set of default text, *Page*, and icons for both the list view per page and the details view per page.

Below is an example code that shows how to change the default icons for the Android template that is available in the Android toolkit.

 **Note:** The framework supports icons of 100 by 100 pixels in size.

Add an *AndroidIconsProvider* class with the new icons

```
public class AndroidIconsProvider extends IconsProvider
{
    // list and details page images
    private static final String LIST_IMAGE_PATH = "res/images/list.png";
    //$NON-NLS-1$
    private static final String DETAILS_IMAGE_PATH =
    "res/images/details.png"; //$NON-NLS-1$

    private Bundle bundle;
    public AndroidIconsProvider()
    {
        this.bundle = Activator.getDefault().getBundle();
    }
    @Override
    public Image getListIcon()
    {
        URL url = this.bundle.getEntry(LIST_IMAGE_PATH);
        ImageDescriptor imageDescriptor =
        ImageDescriptor.createFromURL(url);
        Image image = imageDescriptor.createImage();

        return image;
    }
    @Override
    public Image getDetailsIcon()
    {
        URL url = this.bundle.getEntry(DETAILS_IMAGE_PATH);
        ImageDescriptor imageDescriptor =
        ImageDescriptor.createFromURL(url);
        Image image = imageDescriptor.createImage();
        return image;
    }
}
```

Inside *AndroidList* class overrides the *setContext* method which passes the new icons from the template to the pattern.

```

@Override
    public void setContext (EnumMap<TemplateConstants, Object> context)
    {
        super.setContext(context);

        if (context == null)
        {
            Logger.logWarning("AndroidList template will display
default icons and texts because context argument is null.");
            return;
        }

        Object pattern = context.get(TemplateConstants.PATTERN);
        if (pattern == null)
        {
            Logger.logWarning("AndroidList template will display
default icons and texts because ListPattern is null.");
            return;
        }

        if (pattern instanceof ListPattern)
        {
            ListPattern listpattern = (ListPattern)
context.get(TemplateConstants.PATTERN);

            listpattern.setIconsProvider(new
AndroidIconsProvider());
            listpattern.setLabelsProvider(new
AndroidLabelsProvider());
        }
        else
        {
            Logger.logWarning("AndroidList template will display
default icons and texts because pattern type is not instanceof
ListPattern.");
        }
    }

```

Implementing a New Environment

An environment describes the target runtime platform (toolkit) for a generated application. The environment in which you generate code for an application must closely match the target runtime platform.

The framework supplies its own extension point from which you can derive new extensions for environments.

You can create a new environment by extending the specific extension point, for example, **com.sap.example.eclipse.framework.environment**, available in the framework.

The result of each new environment is the generated environment class file, which you can extend. The generated code for the environment class is commented and has placeholders for you to add your own code.

When you create a new environment, it is added to the list of environment in the framework.


Prerequisites

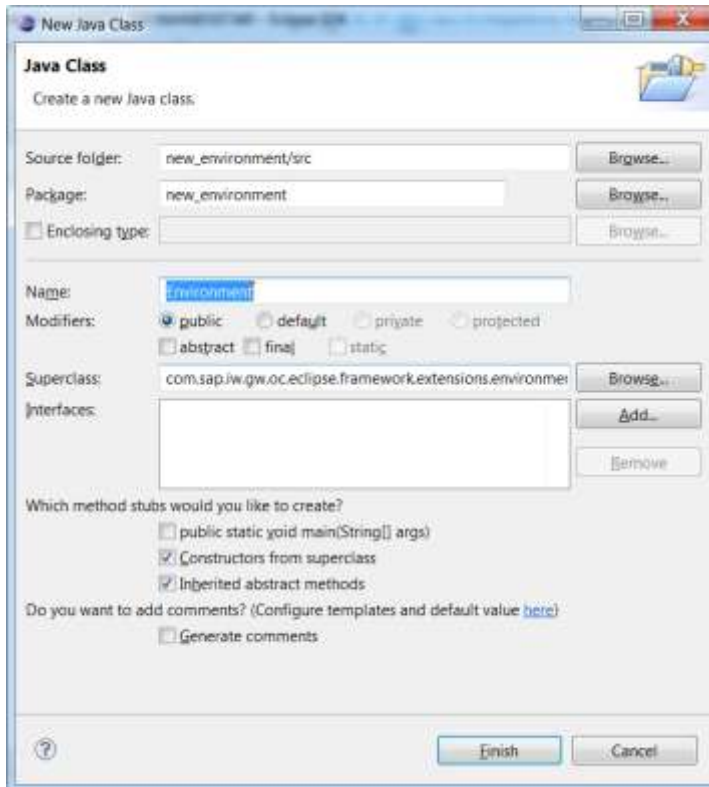
Make sure that you have the following:

A plug-in project in Java.

The file, *Manifest.mf*, or *plug-in.xml* is available in your project.

To create a new environment:

1. Optionally create a new Plug-in Project in Java if you have not done so.
2. Double click the file, **Manifest.mf**, under META-INF in your new plug-in project.
3. Open the Extensions tab, and click **Add** under All Extensions. The Extension Points Selection dialog displays a list of all the extensions points.
4. In Extension Points filter, enter the term, **environment*, to search for all extension points containing the search term.
5. Select the extension point, **com.sap.iw.gw.oc.eclipse.framework.environment**, from the list and choose **Finish**. A new extension element with the same name as the extension point is created.
6. Right click the new extension element, select **New**, and then choose **Environment**.
7. Enter the required values for the attributes with asterisks:
 -  Recommendation: We recommend that you save your changes after modifying the value of each attribute.
 - ID: A unique ID for the environment.
 - Environment: The path of the new environment class to be generated.
 - displayName: A translatable name that will be used in UI representation of this environment in the wizard.
 - description: Descriptive text for the new environment.
 - icon: The path of a graphic file that will visually represent the environment in the wizard.
8. Click the attribute, **Environment**. The New Java Class dialog opens.



9. Enter a name for your Environment class.
The Superclass points to the Extension Point specified for the new environment class.
The selected modifier is Public, and the selected method stubs are constructors from superclass and inherited abstract methods.
10. Click **Finish**, and then click **Save** in the Eclipse main menu.
Code is generated for the new environment in the specified environment class.

Generating a Starter Application from the New Environment

You can create a fully functional starter application based on a template that references your new environment, as the new environment is added to the Starter Application wizard of the framework.

Generating a Proxy from a New Environment

You can create a fully functional proxy based on your new environment, as the new environment is added to the Proxy wizard of the framework.

1. To run and test your new environment, select **Run** from the main menu and choose **Run**.
2. Open the Proxy Wizard of the framework, specify the target environment, provide the required entries, and then generate the proxy.

Extending the New Environment

The environment receives a template type; *Proxy*, or *Starter Application*. Based on this template type, it presents a corresponding user interface (UI) from the framework plug-in.

In the example below, the method creates a specific UI control for the environment. This method returns an environment UI that corresponds to the given template type.

```
public EnvironmentUi createCompositeUI(TemplateType templateType)
throws EnvironmentException
```

- From the Java SE Environment:

```
@Override
public EnvironmentUi createCompositeUI(TemplateType templateType)
throws EnvironmentException
{
    switch(templateType)
    {
        case Proxy:
            this.envUi = new ProxyEnvironmentUi();
            break;
        case Starter_Application:
            this.envUi = new StarterAppEnvironmentUi();
            break;
        default:
            this.envUi = null;
    }
    return this.envUi;
}
```

Recommendation: Each environment will implement an interface, and only that interface will be exposed in the exported packages.

- From the IJavaSeModel interface:

```
public interface IJavaSeModel
{
```

- Returns the name of a project.

```
public String getProject();
```

- Returns the name of the main class of the project.

```
public String getMainClass();
```

- Returns the package name in the project.

```
public String getPackage();
```

- Returns the output path of the generated file.

```
public String getOutputPath();
```

- Creates a new project for a Java Starter Application template.

```
public void createNewProject() throws EnvironmentException;
```

- Updates the project to a proxy project, by adding the Client library, **.project** and **.classpath** files.

```
public void updateProxyProject() throws EnvironmentException;  
}
```

Troubleshooting

General

Using Multiple Types of the Same Gateway Service

Issue: When running the starter application or the proxy wizard, you cannot choose whether to generate an application using an OData-compliant or Sdata-compatible Gateway service.

For example, `http://<gateway host>:<port>/sap/opu/sdata/sap/Z_SV_ZVD11`. Or, `http://<gateway host>:<port>/sap/opu/odata/sap/Z_SV_ZVD11`

By default, a wizard chooses the OData-compliant service (as this is the recommended format), though a service may be available in both formats.

Solution: You must regenerate the proxy or starter application using the URL of the service type. For more information see, *Viewing your Search Results* on page 14.

Troubleshooting Java Platform Standard Edition Toolkit

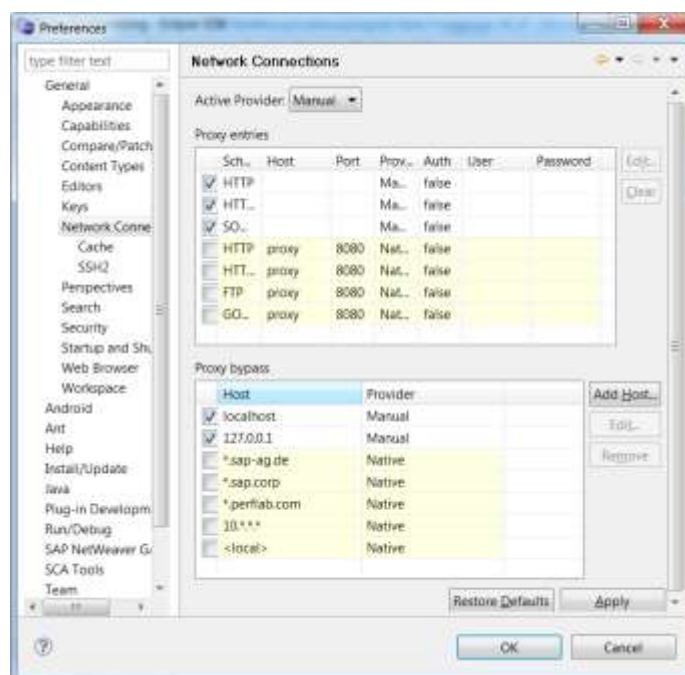
Network Connections

Issue: When Eclipse is configured to use Native active provider in Network Connections, you cannot use short host name, such as, `vmw.example.com`.

Eclipse attempts to connect through the proxy, even if the server is configured in the bypass list in Windows.

Solution: Either use the fully qualified domain name (FQDN) of the SAP NetWeaver Gateway server or configure the Network Connections as follows:

1. From the Eclipse menu, go to **Window** → **Preferences** → **General** → **Network Connections**.
2. Select **Manual** in the **Active Provider**.
3. In the proxy entries table, select **Add Host** and provide the SAP NetWeaver Gateway server details, including the host name, port, username and password.



Proxy Generation Errors

Issue: Some SAP services that are OData compliant may contain entities, complex types, or fields that are similar to Java reserved keywords, such as, *return*, *class*, *public*, and many more.

When the Proxy Generation wizard comes across these occurrences, it creates proxy files with incorrect content, such as, *return* as the entity type name and *abc-dfg* as a property name.

Solution: After the proxy is generated, manually fix the compilation errors by renaming the generated code to use legal Java names. For example, *return1*, *class1*, *public1*.

Starter Application Wizard Template Error

Issue: When you choose to create a new starter application for the Java SE environment, after providing the entries for your application, you can go back to switch to creating a PHP application, however, the wizard presents you with the Finish option.

Selecting Finish results in an error.

Solution: Do not choose Finish at this point as the wizard requires you to specify the entries for the List/Details Application template.

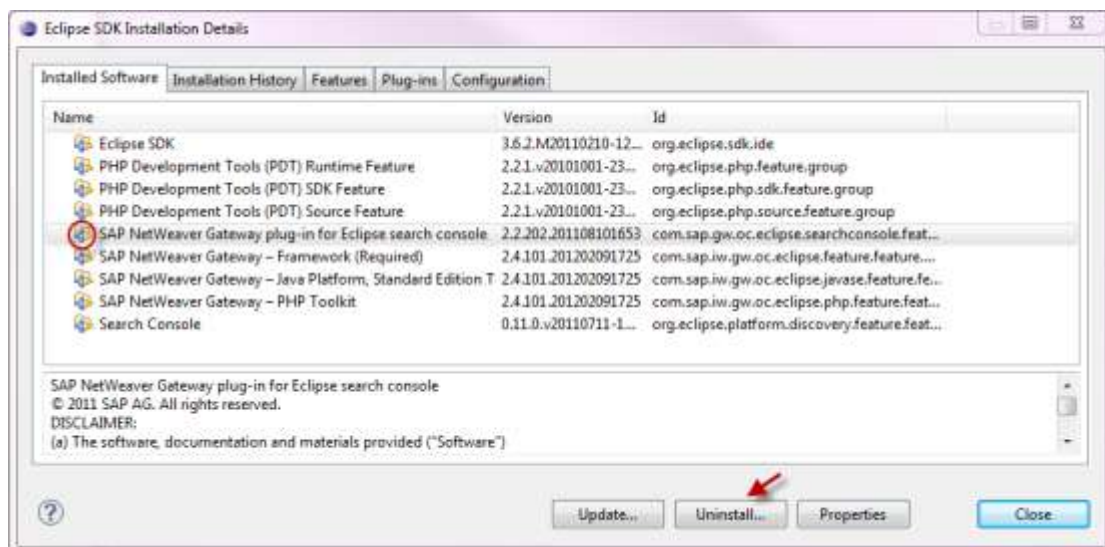
Search Provider Errors

Issue: The Search Console does not display the search results. This occurs when there is a conflict in the installed versions of the search provider.

Older version of the search provider (version 2.2.202) has not been overwritten, causing a conflict with the newer version of the search provider.

Solution:

Uninstall the older version of the search provider (version 2.2.202) and restart Eclipse.



To uninstall, see **Uninstalling**, on page 16.

The newer version of the search provider is an integral part of the installed framework plug-in.

PHP Toolkit Troubleshooting

PHP Duplicated Class Error


Issue: The error, *Duplicated Class Error*, displays at runtime of the PHP application because the name of the specified SAP service is the same as one of its entity types. This results in the creation of two PHP classes with the same name. Class names are case insensitive.

Solution: Open the generated class file (PHP proxy file), for example, `.../services/<service_name>.php`, and change the service class name that has been defined inside it. You can identify this class as the one that extends the *ObjectContext* class.

For example, add the underscore character to the service name, `'_SERVICE'`.

Note that you must manually update all the references to the service class in the source code, to point to the new class name.

Find the only reference to the service class in the file, *service.php*, for initializing the proxy.

-  **Note:** Renaming the entity type class name is not recommended since it may break the generated proxy code.

Generated Class Names or Variable Names are Reserved for PHP

Issue: The selected SAP service may contain entity, collection, service or property names that are reserved words in the PHP programming language such as. *'return'*, or contain illegal characters such as *'*'*.

Solution: Open the generated class file (PHP proxy file), for example, `.../services/<service_name>.php`, and rename all the class and variable names which are illegal in PHP, and then update all the references to such items within the proxy and the application code.

Proxy Generation Failed

Issue: Generating PHP starter application or proxy failed, and the Eclipse error log indicates, *'Proxy Generation Failed'* error.

Solution: Verify the installation and configuration of the OData SDK for PHP. Follow the procedures in the *'Installation and Configuration'* section of its user guide located at the *"doc/User_Guide.htm"* file in the file system location for the downloaded SDK.

On Mac machines, also follow the instructions below:

- In the OData SDK for PHP code, change all the `'\'` character occurrences to `'/'` character in the *'require_once'* block of the *"Common/ACSUtil.php"* file.
For more information, refer to the open issue at:
odataphp.codeplex.com/workitem/11143.
- Make sure the OData SDK for PHP is configured in the *php.ini* file on your local PHP server as following:

```
include_path = "<PHP_Libraries_Include_Path>:<OData_SDK_Path>"
;OData SDK for PHP Library Path
Odataphp_path = "<OData_SDK_Path>"
extension=curl.co
extension=xsl.co
;Defines the default timezone used by the date functions
date.timezone = <Local_Timezone_As_In:
http://www.php.net/manual/en/timezones.php>
```
- Regenerate the PHP proxy/starter application, and rename the PHP proxy file generated, for example, `.../services\<service_name>.php` to `<service_name>.php`, and then relocate it to the folder, *'services'* in the project.

PHP Runtime Issues

Missing Files in Linux or Mac Operating Systems

Issue: The following PHP error occurs at runtime in the UNIX/LINUX or Mac server machine:

For example:

```
Warning: require_once(Resource\Messages.php) [function.require-
once]: failed to open stream: No such file or directory
in /Applications/XAMPP/xamppfiles/htdocs/odataphp/Common/ACSUtil.php on line 17
```

```
Fatal error: require_once() [function.require]: Failed opening
required 'Resource\Messages.php'
(include_path='./Applications/XAMPP/xamppfiles/lib/php:/Applications/XAMPP/xamppfiles/lib/php/pear:/Applications/XAMPP/xamppfiles/htdocs/odataphp/')
in /Applications/XAMPP/xamppfiles/htdocs/odataphp/Common/ACSUtil.php on line 17
```

The error indicates missing files that are required by the file,
<OData_SDK_Path>/Common/ACSUtil.php

Solution: Open the OData SDK for PHP code, and change all occurrences of the character '`\`' character to '`/`' in the '`require_once`' block in the file, .../Common/ACSUtil.php

For more information, refer to the open issue: odataphp.codeplex.com/workitem/11143 .

Missing OData SDK Files

Issue: The following PHP error occurs at runtime:

For example:

```
Warning: require_once(Context/ObjectContext.php)
[function.require-once]: failed to open stream: No such file or
directory in
/Applications/XAMPP/xamppfiles/htdocs/Test1/services/
WFODCPROCESSING.php on line 27
```

```
Fatal error: require_once() [function.require]: Failed opening
required 'Context/ObjectContext.php'
(include_path='./Applications/XAMPP/xamppfiles/lib/php:/Applications/XAMPP/xamppfiles/lib/php/pear') in
/Applications/XAMPP/xamppfiles/htdocs/Test1/services/
WFODCPROCESSING.php on line 27
```

The error indicates that there are missing files for OData that are required from the generated proxy.

Solution: Make sure that the OData SDK for PHP is configured in the '`include_path`' parameter of the file, `php.ini`, in your PHP server as follows:

- For UNIX/LINUX:
`include_path="<PHP_Libraries_Include_Path>:/<OData_SDK_Path>"`
- Mac OS X:
`include_path="<PHP_Libraries_Include_Path>:/<OData_SDK_Path>"`
- Windows:
`include_path="<PHP_Libraries_Include_Path>;<C:\OData_SDK_Path>"`

Undefined CURL Library Functions

Issue: The following PHP error occurs at runtime:

For example:

```
Fatal error: Call to undefined function curl_init() in
/Applications/XAMPP/xamppfiles/htdocs/odataphp/WebUtil/
HttpRequest.php on line 124
```

The error indicates that there are undefined CURL library functions.

Solution: Make sure that the CURL library extension is located in the same folder as the PHP server, and the CURL library extension is uncommented in the file, *php.ini*, in your PHP server as follows:

- For UNIX/LINUX:
extension="curl.so"
- For Mac OS X:
extension=curl.co
- For Windows:
extension=php_curl.dll

Generated Proxy or Starter Application Exception

Issue: Sometimes, a ProxyGenerationException occurs when generating a PHP proxy or starter application.

Solution: Verify that you have correctly located only the files from the framework folder into the folder, *odataphp*, and not the entire folder for the framework.

Service Metadata Document from your Local File System Causes Runtime Error

Issue: When you generate an application using a service metadata document (XML) file from the file system, you get the following error at the runtime of the application:

```
Error Occured!
Error occurred during request for
<path_to_metadata_xml_file>/ExtensibleElementUpdateCollection: Could not resolve host:
<path_to_metadata_xml_file>; Host not found
```

Solution: Open the project of the generated application, and edit the file, *service.php*, as follows:

Find the parameters SERVICE_URL, and SAP_CLIENT, and set the correct URL and SAP Client values.

Before changes:

```
define('SERVICE_URL','<path_to_metadata_xml_file >');
define('SAP_CLIENT','${GATEWAY_CONFIGURATION.getClient()}');
//Leave empty for default SAP client
```

After changes:

```
define('SERVICE_URL','<service_url>');
define('SAP_CLIENT','<sap_client_number>');
//Leave empty for default SAP client
```

Troubleshooting the Android Toolkit

Android Runtime Issues

Generated Proxy Configuration for External Network Connectivity

Issue: At runtime, when you attempt to connect to an external network other than your local network, you get the exception, *UnknownHostException* message.

Solution: In Eclipse, open the file, *SDMConnectivityHelper.java* class in the helpers package of the generated service proxy, add the following, and then redeploy the application:

```
this.mPreferences.setStringPreference(ISDMPreferences.SDM_CONNECTIVITY_PROXY_HOST, "<your proxy host>");  
  
this.mPreferences.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_PROXY_PORT, <your proxy port>);
```

Starter Application Timed Out Error at Runtime

Issue: In some cases, you may get the following error at runtime: *java.io.IOException: JSON: expected character; found EOF*.

This error occurred because the SUP server timed out (60 seconds).

Solution: From the generated package, open the file, *LoginActivity.java*, and add the *\$top* parameter to the query request in order to restrict the number of items retrieved.

For example:

```
ODataQuery query = application.getService().getBookingCollectionQuery();  
query.addParameter("top", "10");
```

File Out of Sync Error

Issue: When running the Android Toolkit in versions of Eclipse that are higher than Eclipse 3.7.0, you may get the following error when you generate the application: *File out of synch*.

Or, at runtime, in the emulator, the following error displays: *ActivityManager: Error type 3, ActivityManager: Error: Activity class {xx.xx/xx.xx.SplashScreen} does not exist*,

Solution: Open **Window** → **Preferences** → **General** → **Workspace**, and select **Refresh on access**, before you regenerate the project.



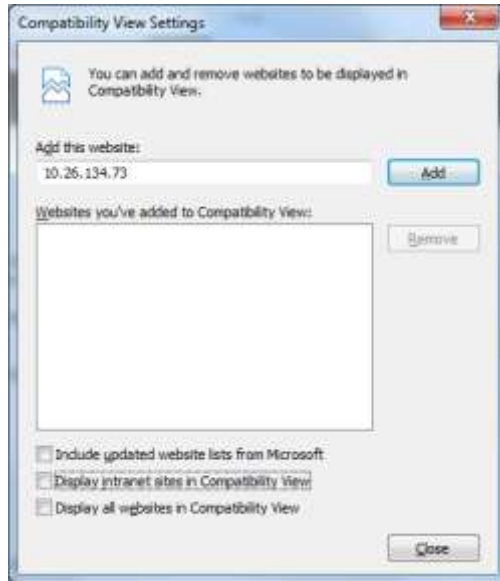
Troubleshooting the HTML5 Toolkit

JSON Parse Exception in Internet Explorer 9 (IE9)

Issue: When running the generated application using IE9 RC1, with an SAP NetWeaver Gateway service supporting JSON, the following error displays:

"Unable to get value of the property 'parse': object is null or undefined".

Solution: Disable displaying sites in compatibility view in your IE9 Web browser by selecting **Page** → **Compatibility View Settings** and remove selection for the option **Display internet sites in Compatibility View**.



Terminology

Custom Application	The application that you design and develop in order to interface with an SAP service through SAP NetWeaver Gateway using the SAP NetWeaver Gateway Plug-in for Eclipse.
List Application	The starter application that you generate based on a predefined design and flow, to consume an SAP service through SAP NetWeaver Gateway using the SAP NetWeaver Gateway Plug-in for Eclipse.
Target Extension	A specific environment for which you generate native code using the SAP NetWeaver Gateway Plug-in for Eclipse.
Supported Extension	A specific environment installed on top of SAP NetWeaver Gateway Plug-in for Eclipse to facilitate code generation for applications to be used in that environment.
Gateway service	An OData compliant service available in SAP NetWeaver Gateway for retrieving SAP data and business objects from SAP backend systems.
Gateway server	The host machine running SAP NetWeaver Gateway.
Starter application	An automatically generated application to help you familiarize yourself with how to consume Gateway services through SAP NetWeaver Gateway.
Search Console	An Eclipse plug-in containing a search provider for discovering and exploring Gateway services.