Implementing a Callable Object for Background Execution
## Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Text</td>
<td>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.</td>
</tr>
<tr>
<td>Example text</td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles.</td>
</tr>
<tr>
<td>EXAMPLE TEXT</td>
<td>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.</td>
</tr>
<tr>
<td>Example text</td>
<td>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.</td>
</tr>
<tr>
<td>Example text</td>
<td>Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td>&lt;Example text&gt;</td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td>EXAMPLE TEXT</td>
<td>Keys on the keyboard, for example, <em>F2</em> or ENTER.</td>
</tr>
</tbody>
</table>

## Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caution</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
<tr>
<td>Recommendation</td>
<td></td>
</tr>
<tr>
<td>Syntax</td>
<td></td>
</tr>
</tbody>
</table>
## Contents

Business Scenario .................................................................................................................. 1  
About This Document ........................................................................................................... 1  
Applicable Releases ............................................................................................................. 1  
Disclaimer .............................................................................................................................. 1  
The Step-By-Step Solution ..................................................................................................... 2  
  Create a Java Development Component ............................................................................ 2  
  Implementing a Background Callable Object .................................................................... 3  
  Creating and Deploying a J2EE Library ............................................................................. 10  
  Expose a Java Class as a Callable Object ........................................................................ 11  
  Test and Activate the Background Callable Object ......................................................... 13  
Appendix .................................................................................................................................. 1
Business Scenario

In Guided Procedures (GP) you can encapsulate any function that does not expose a user interface as a callable object for background execution. For example, if you need to implement a certain logic that should remain hidden from the user, you can create a Java class that is executed transparently in background mode at runtime.

In this tutorial, you will learn how to implement a simple callable object that retrieves user details for a given user ID.

About This Document

The implementation of a background callable object to make it available for use in a process has the following phases:

1. You create and configure a Java development component (DC).
2. You implement your background callable object in the above component.
3. You create a J2EE library and deploy it to the SAP Web Application Server.
4. You expose the class as a callable object in the GP design time.
5. Finally, you test and activate the object.

⚠️

This How To guide explains how to implement a background callable object using SAP NetWeaver 2004s SPS07 APIs. It differs slightly from the APIs available with SPS06. If you are using SAP NetWeaver SPS06, see SAP Note 924776 and the notes in the text that describe the differences from SPS06.

Applicable Releases

This tutorial is compatible with the following release “Beginning with SAP NetWeaver 2004s SPS07”.

Disclaimer

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.
The Step-By-Step Solution

Create a Java Development Component

Using this procedure, you create a Java development component (DC) project, in which you create the callable object.

To use the interfaces required by Guided Procedures (GP) for background callable objects, you must also configure the project by adding the appropriate DCs.

Your development configuration must contain the following DCs:

- caf/eu/gp/api
- com.sap.exception
- com.sap.security.api.sda
- tc/logging

1. Open the Java Perspective of the SAP NetWeaver Developer Studio and choose File → New → Project.…
2. Select Development Component → Development Component Project. Choose Next.
3. Select the software component where you want to create your DC. Choose Next.
4. Enter a name for the development component, for example, bckg_co. Select the appropriate domain. For the type, select Java. Choose Finish.
5. Go to Window → Show View → Other… → Java → Java DC Explorer.
6. Expand the tree of the development component you created, select DC MetaData → DC Definition → Used DCs, and open the context menu.
7. Choose Add Used DC….
8. Browse your development configuration. Select DC caf/eu/gp/api and its public part external.

⚠️ In SAP NetWeaver 2004s SPS06, you must select the caf/eu/gp/api DC, as the public part is not available.

9. For the dependency type, select Build Time.
11. Repeat the previous step to declare build-time dependencies to the following DCs:

   - com.sap.exception
   - com.sap.security.api.sda
   - tc/logging
Implementing a Background Callable Object

The Java class to be used as a callable object for background execution must implement the interface com.sap.caf.eu.gp.co.api.IGPBackgroundCallableObject. It enables the execution of the class within the Guided Procedures (GP) framework.

⚠️

In SAP NetWeaver 2004s SPS06 the interface is called com.sap.caf.eu.gp.co.api.IBackgroundColorableObject.

The most important aspects of the implementation include:

- Defining input parameters, output parameters, and result states in method getDescription()
- Setting the output parameters and completing the execution in method execute()
- Implementing localization for texts
- Implementing exception handling

You can find the complete code of the background callable object in the Appendix.

12. Create the Java class:
   a. In Package Explorer perspective, expand the tree of your Java DC project and select src/packages. Choose File → New → Package.
   b. Enter a name for your package, for example, com.examples.bckgco. Choose Finish.
   c. In your project tree, select the package you created and choose File → New → Class.
      • Enter a name for the class, for example, UserDetailsCallableObject.
      • To define that the class implements IGPBackgroundCallableObject, choose Add… next to Interfaces, and search for the interface.
      • Make sure only the option Inherited abstract methods is selected.
   d. Choose Finish. The class opens in the Java editor.

13. Edit method getDescription():
This method returns the technical description of the callable object, which contains metadata about the object’s input, output, and configuration parameters, as well as about its result states.

To define the callable object:
   a. Retrieve the existing root structures for the input and output parameters using an instance of the com.sap.caf.eu.gp.co.api.IGPTechnicalDescription interface. It is instantiated using com.sap.caf.eu.gp.co.api.GPCallableObjectFactory.

You can define sub-structures and attributes for structures using the interfaces IGPStructureInfo and IGPAttributeInfo.

⚠️

In SAP NetWeaver 2004s SPS06 these classes interfaces are as follows:
- com.sap.caf.eu.gp.co.api.ITEchnicalDescription
In this example, you define a single input parameter for the user ID in the root structure. For the output parameters, you create a structure where you define the attributes for the user details that you retrieve – first and last name, telephone and fax numbers (see the coding example below).

The technical names that you use for the parameters of the callable object may only contain numbers, letters of the Latin alphabet, and the underscore character.

```java
public IGPTechnicalDescription getDescription(Locale originalLocale) {
    try {
        //create technical description instance
        IGPTechnicalDescription technicalDescription =
            GPCallableObjectFactory.newTechnicalDescription(
                "CO_NAME",
                "CO_DESCRIPTION",
                resourceAccessor,
                originalLocale);

        //get root structure for input parameters
        IGPStructureInfo input = technicalDescription.getInputStructureInfo();

        //define attributes for input parameters
        IGPAttributeInfo userId =
            input.addAttribute("UserID", IGPAttributeInfo.BASE_STRING);
        userId.setMultiplicity(IGPAttributeInfo.MULTIPLICITY_1_1);

        //get root structure for output parameters
        IGPStructureInfo output =
            technicalDescription.getOutputStructureInfo();

        //add a sub-structure for the output
        IGPStructureInfo user = output.addStructure("User");

        //define attributes in output sub-structure
        user.addAttribute("firstName", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("lastName", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("phone", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("fax", IGPAttributeInfo.BASE_STRING);
    }
```
b. Set the result states for the callable object.
The example below defines a result state that indicates successful execution. The interface com.sap.caf.eu.gp.co.api.IGPCOResultStateInfo is used for the result state definition.

⚠️
In SAP NetWeaver 2004s SPS06 this interface is called com.sap.caf.eu.gp.co.api.IResultStateInfo.

```java
public IGPTechnicalDescription getDescription(Locale originalLocale) {
    try {
        //add a result state for successful execution
        IGPCOResultStateInfo success = technicalDescription.addResultState("Success");
        //add result state description
        success.setDescriptionKey("Success_desc");
        return technicalDescription;
    } ...
}
```

14. Edit method execute():
In this method you implement the runtime behavior of the callable object.

   a. You use an instance of com.sap.caf.eu.gp.co.api.IGPEXecutionContext to retrieve runtime information – for example, to add values to the attributes you defined in method getDescription().

⚠️
In SAP NetWeaver 2004s SPS06 this interface is called com.sap.caf.eu.gp.co.api.IExecutionContext.

```java
public void execute(IGPEXecutionContext executionContext)
    throws GPTechnicalCallableObjectException {
    try {
        //retrieve the runtime representation of the input structure
        IGPStructure input = executionContext.getInputStructure();
        //retrieve the value of the input parameter
        String userId = (String) input.getAttributeAsSting("UserID");
```
//retrieve the runtime representation of the output structure
IGPStructure output = executionContext.getOutputStructure();
IGPStructure user = output.addStructure("User");

try {
    //find user in user management
    IUser umeUser =
        UMFactory.getUserFactory().getUserByLogonID(userId);
    //set values to output parameters
    user.setAttributeValue("firstName", umeUser.getFirstName());
    user.setAttributeValue("lastName", umeUser.getLastName());
    user.setAttributeValue("phone", umeUser.getTelephone());
    user.setAttributeValue("fax", umeUser.getFax());

    //set result state
    executionContext.setResultState("Success");

}...

//complete object execution
executionContext.processingComplete();
}

c. Define a result state that is reached when the object is successfully executed.

public void execute(IGPExecutionContext executionContext) throws GPTechnicalCallableObjectException { 
    try {
        ...
        try {
            ...
            //set result state
            executionContext.setResultState("Success");
            ...
        }...
    }
}

c. Finally, complete object execution by calling method processingComplete() of the execution context.

public void execute(IGPExecutionContext executionContext) throws GPTechnicalCallableObjectException {
    try {

15. Implement process exception handling:
   a. Add a process exception for the background callable object. You do this in the
getDescription() method.

```java
public IGPTechnicalDescription getDescription(Locale originalLocale) {
    try {
        ...
        //add a process exception for the callable object
        technicalDescription.addProcessException("EXCEPTION_NO_USER_FOUND");
        return technicalDescription;
    }
}
```

b. Implement a catch clause in the execute() method so that a process exception occurs if there
is a UME exception.

```java
public void execute(IGPExecutionContext executionContext)
    throws GPTechnicalCallableObjectException {
    String userId = null;
    try {
        ...
        try{
            ...
        } catch (UMException e) {
            executionContext.setProcessException("EXCEPTION_NO_USER_FOUND");
        }
        executionContext.processingComplete();
    }
}
```
16. Implement technical exception handling:
   c. Create an instance of com.sap.tc.logging.Location for logging error messages. For more information, see the API documentation at http://www.sdn.sap.com/irj/sdn/javadocs.
   d. In method getDescription(), implement the catch clause of the try-catch block to trace and log an invocation exception. In case of an error, the method should return null.

   ![Error](image)
   
   This exception is thrown if the technical names contain invalid characters. Therefore, it should not occur unless the naming rules are violated.

   ![Warning](image)
   
   In SAP NetWeaver 2004s SPS06 the exception class is called com.sap.caf.eu.gp.co.api.InvocationException.

```java
public IGPTechnicalDescription getDescription(Locale originalLocale) {
    try {
        ...
    } catch (GPInvocationException e) {
        logger.logT(
            Severity.ERROR,
            Category.APPLICATIONS,
            "Incorrect technical name");
        logger.traceThrowableT(
            Severity.ERROR,
            "Exception while creating technical description: ",
            e);
        return null;
    }
}
```

   e. In method execute(), implement the catch clause so that a GP technical exception is thrown if an error occurs. You can localize the detailed messages for each individual error.

   ![Warning](image)
   
   In SAP NetWeaver 2004s SPS06, the exception classes used in the below example are as follows:

   - com.sap.caf.eu.gp.co.api.InvocationException (instead of com.sap.caf.eu.gp.co.api.GPInvocationException)
   - com.sap.caf.eu.gp.co.api.EngineException (instead of com.sap.caf.eu.gp.co.api.GPEngineException)
   - com.sap.caf.eu.gp.co.api.TechnicalException (instead of com.sap.caf.eu.gp.co.api.GPTechnicalCallableObjectException)

```java
public void execute(IGPExecutionContext executionContext)
```
throws GPTechnicalCallableObjectException {
    String userId = null;
    try {

        ... 

    } catch (GPInvocationException e) {
        throw new GPTechnicalCallableObjectException (logger,
        resourceAccessor, "ERROR_PARAMETERS", e);
    } catch (GPEngineException e) {
        throw new GPTechnicalCallableObjectException (logger,
        resourceAccessor, "INTERNAL_ERROR", e);
    }
}

17. Implement localization:
The names you defined for the callable object parameters can be used automatically as keys for localized messages that are displayed at runtime. You can also set additional keys, as shown in the above example for defining a result state description.

To implement localization:

a. In the same package, create a new class that extends com.sap.caf.eu gp.co.api.GPStandardResourceAccessor – for example, UserDetailsResourceAccessor.

    In SAP NetWeaver 2004s SPS06, the class is called com.sap.caf.eu gp.co.api.StandardResourceAccessor.

b. In your resource accessor class, add the constructor from the super class, as shown in the example below.

```java
public class UserDetailsResourceAccessor extends GPStandardResourceAccessor {

    /**
     * @param bundleName
     */
    public UserDetailsResourceAccessor(String bundleName) {
        super(bundleName);
    }
}
```

c. In the same package, create a text file with extension properties – for example, UserDetails.properties. Enter all texts that you want to localize as key-value pairs in the format shown in the below example.

```
CO_NAME=User Details
```
d. In class UserDetailsCallableObject, instantiate the resource accessor class that you created. Enter the name of the resource bundle as a fully qualified name including the package, as in the example below.

```java
public class UserDetailsCallableObject implements IGPBackgroundCallableObject {
    StandardResourceAccessor resourceAccessor =
        new UserDetailsResourceAccessor("com.examples.bckgco.UserDetails");

    ...
}
```

e. Save the files and build your project. You have implemented a Java class that can be exposed as a callable object in GP.

**Creating and Deploying a J2EE Library**

To be able to deploy your background callable object and access it from Guided Procedures, you must package it as a J2EE library and deploy it on the Java server.

For this purpose, you must:

- Create a public part in your existing Java development component (DC) project including the background callable object class.
- Create a J2EE library DC that wraps the above class.
- Deploy the library to the Java server.

As a prerequisite for this procedure, you must have configured the connection to the Java server in the Window → Preferences → SAP J2EE Engine menu of the SAP NetWeaver Developer Studio.
18. In *Java DC Explorer*, expand the bckg_co DC tree and select DC Meta Data → Public Parts.
   f. Open the context menu and choose New Public Part.
   g. Enter a name for the public part, for example COImpl, and select the option Can be packaged into other build results (e.g. SDAs).
   h. Choose Next. Select entity type Java Package Tree. Under Select Entities, expand the tree that appears, and select the uppermost element com.
   i. Make sure that option as Class is selected for Add Entity to Public Part. Choose Finish.

19. Choose File → New → Project:
   j. Select Development Component → Development Component Project.
   k. Choose Next. Select the software component where you want to create your DC.
   l. Choose Next. Enter a name for the development component, for example, apibckg. Select the appropriate domain. For the type, select J2EE Server Component → Library.
   m. Choose Finish.

20. In *Java DC Explorer*, expand the apibckg DC tree, and select DC Meta Data → DC Definition → Used DCs.
   n. Open the context menu and choose Add Used DC. Browse to select the COImpl public part of the bckg_co DC. For the dependency type, choose Build Time.
   o. Repeat the procedure to add Build Time, Deploy Time, and Run Time dependencies for the following DCs:
      - caf/eu/gp/api (public type external)
      - com.sap.security.api.sda

21. Select apibckg DC, open its context menu, and choose Development Component → Build. Check for warnings or errors in the General User Output view.

22. From the context menu of the apibckg DC, choose Development Component → Deploy. Check for warnings or errors in the Deploy Output View.

**Expose a Java Class as a Callable Object**

As a prerequisite, you must hold the portal role GP Business Expert or the relevant permissions for working with the Guided Procedures (GP) design time.
23. Launch your portal and open Guided Procedures → Design Time.

24. Choose Create Callable Object from the contextual panel.

25. Enter the following data:
   - Type – select Background Execution
   - Name – for example, User Details
   - Description – for example, Retrieve user details in the background
   - Original Language – select English
   - Location – choose any folder in the Gallery

26. Choose Next to open the Define Object screen.
   - For Container Type, select Library.
   - For Container, select the J2EE library that you created and deployed to the application server.
   - For Implementation Class Name, enter the fully qualified Java name of your implementation class.

27. Choose Next to display the Define Input screen. The input parameter UserID that you defined in the implementation is displayed here.
28. Choose Next to display the Define Output screen. The structure with the output parameters defined in the implementation is displayed.

29. Choose Next, and then Finish and Open.

Test and Activate the Background Callable Object

30. Create user John Smith with user ID smith in the User Administration console of the portal.
   Define the following data for this user:
   - Telephone number
   - Fax number

31. Go back to the callable object design time and choose (Edit) to switch to edit mode.

32. Open the Test tab page.

33. Enter smith for User ID as shown in the example.
34. Choose *Execute*.

If your implementation was successful, the system executes the callable object and opens the test results view.

In our example the result is *Successful completion*, and the output parameters are displayed.

35. Choose ✅ (*Activate*).
Appendix

UserDetailsCallableObject.java

```java
package com.examples.bckgco;

import java.util.Locale;

import com.sap.caf.eu.gp.co.api.GPCallableObjectFactory;
import com.sap.caf.eu.gp.co.api.GPStandardResourceAccessor;
import com.sap.caf.eu.gp.co.api.IGPBackgroundCallableObject;
import com.sap.caf.eu.gp.co.api.IGPCOResultStateInfo;
import com.sap.caf.eu.gp.co.api.IGPExceptionInfo;
import com.sap.caf.eu.gp.co.api.IGPExecutionContext;
import com.sap.caf.eu.gp.co.api.IGPTechnicalDescription;
import com.sap.caf.eu.gp.exception.api.GPEngineException;
import com.sap.caf.eu.gp.exception.api.GPInvocationException;
import com.sap.caf.eu.gp.exception.api.GPTechnicalCallableObjectException;
import com.sap.caf.eu.gp.structure.api.IGPAttributeInfo;
import com.sap.caf.eu.gp.structure.api.IGPStructure;
import com.sap.caf.eu.gp.structure.api.IGPStructureInfo;
import com.sap.security.api.IUser;
import com.sap.security.api.UMException;
import com.sap.security.api.UMFactory;
import com.sap.tc.logging.Category;
import com.sap.tc.logging.Location;
import com.sap.tc.logging.Severity;

public class UserDetailsCallableObject implements IGPBackgroundCallableObject {

    public static Location logger =
        Location.getLocation(UserDetailsCallableObject.class);

    //initialize the resource accessor class with the resource bundle as a parameter
    GPStandardResourceAccessor resourceAccessor =
        new UserDetailsResourceAccessor("com.examples.bckgco.UserDetails");

```
public IGPTechnicalDescription getDescription(Locale originalLocale) {
    try {

        // create technical description instance
        IGPTechnicalDescription technicalDescription =
            GPCallableObjectFactory.createTechnicalDescription("CO_NAME",
                        "CO_DESCRIPTION",
                        resourceAccessor,
                        originalLocale);

        // get root structure for input parameters
        IGPStructureInfo input =
            technicalDescription.getInputStructureInfo();

        // define attributes for input parameters
        IGPAtributeInfo userId =
            input.addAttribute("UserID", IGPAttributeInfo.BASE_STRING);
        userId.setMultiplicity(IGPAttributeInfo.MULTIPLICITY_1_1);

        // get root structure for output parameters
        IGPStructureInfo output =
            technicalDescription.getOutputStructureInfo();

        // add a sub-structure for the output
        IGPStructureInfo user = output.addStructure("User");

        // define attributes in output sub-structure
        user.addAttribute("firstName", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("lastName", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("phone", IGPAttributeInfo.BASE_STRING);
        user.addAttribute("fax", IGPAttributeInfo.BASE_STRING);

        // add a result state for successful execution
        IGPCOResultStateInfo success =
            technicalDescription.addResultState("Success");
        success.setDescriptionKey("Success_desc");

        IGPExceptionInfo exception =
technicalDescription.addProcessException("EXCEPTION_NO_USER_FOUND");

    return technicalDescription;

} catch (GPInvocationException e) {
    logger.logT(
        Severity.ERROR,
        Category.APPLICATIONS,
        "Incorrect technical name");
    logger.traceThrowableT(
        Severity.ERROR,
        "Exception while creating technical description: ",
        e);
    return null;
}
}

public void execute(IGPExecutionContext executionContext)
    throws GPTechnicalCallableObjectException {
    String userId = null;

    try {

        //retrieve the runtime representation of the input structure
        IGPStructure input = executionContext.getInputStructure();

        //retrieve the value of the input parameter
        userId = (String) input.getAttributeAsString("UserID");

        //retrieve the runtime representation of the output structure
        IGPStructure output = executionContext.getOutputStructure();
        IGPStructure user = output.addStructure("User");

        try {

            //find user in user management
            IUser umeUser =
                UMFactory.getUserFactory().getUserByLogonID(userId);

            //set values to output parameters
            user.setAttributeValue("firstName", umeUser.getFirstName());

    }
user.setAttributeValue("lastName", umeUser.getLastName());
user.setAttributeValue("phone", umeUser.getTelephone());
user.setAttributeValue("fax", umeUser.getFax());

    //set result state
    executionContext.setResultState("Success");
}

} catch (UMException e) {

    //set process exception if user is not found
    executionContext.setProcessException("EXCEPTION_NO_USER_FOUND");
}

    //complete object execution
    executionContext.processingComplete();

} catch (GPInvocationException e) {
    throw new GPTechnicalCallableObjectException(
        logger,
        resourceAccessor,
        "ERROR_PARAMETERS",
        e);
}

} catch (GPEngineException e) {
    throw new GPTechnicalCallableObjectException(
        logger,
        resourceAccessor,
        "INTERNAL_ERROR",
        e);
    }
    }
www.sap.com/netweaver