Ready to Build Your First MiniApp? It's Quick and Easy with the ABAP Workbench!

Alfred Barzewski

Alfred Barzewski joined SAP in 1997 as a member of the ABAP Workbench product management group, where he was responsible for online documentation on Remote Communication, non-SAP access to BAPIs, RFC programming in ABAP, and ABAP development tools. He currently works on documentation for SAP's Web Development platforms, focusing on implementation of reusable demo components for ABAP developers.

(complete bio appears on page 24)

I know that many SAP customers already have installed Workplace clients in pockets of their organizations and that many more intend to do the same in the months ahead. The allure of this product is that it offers users browser-based access to information, applications, and services, across SAP and non-SAP systems alike, in a way that befits each user's unique role within the company. MiniApps are instrumental in this process.

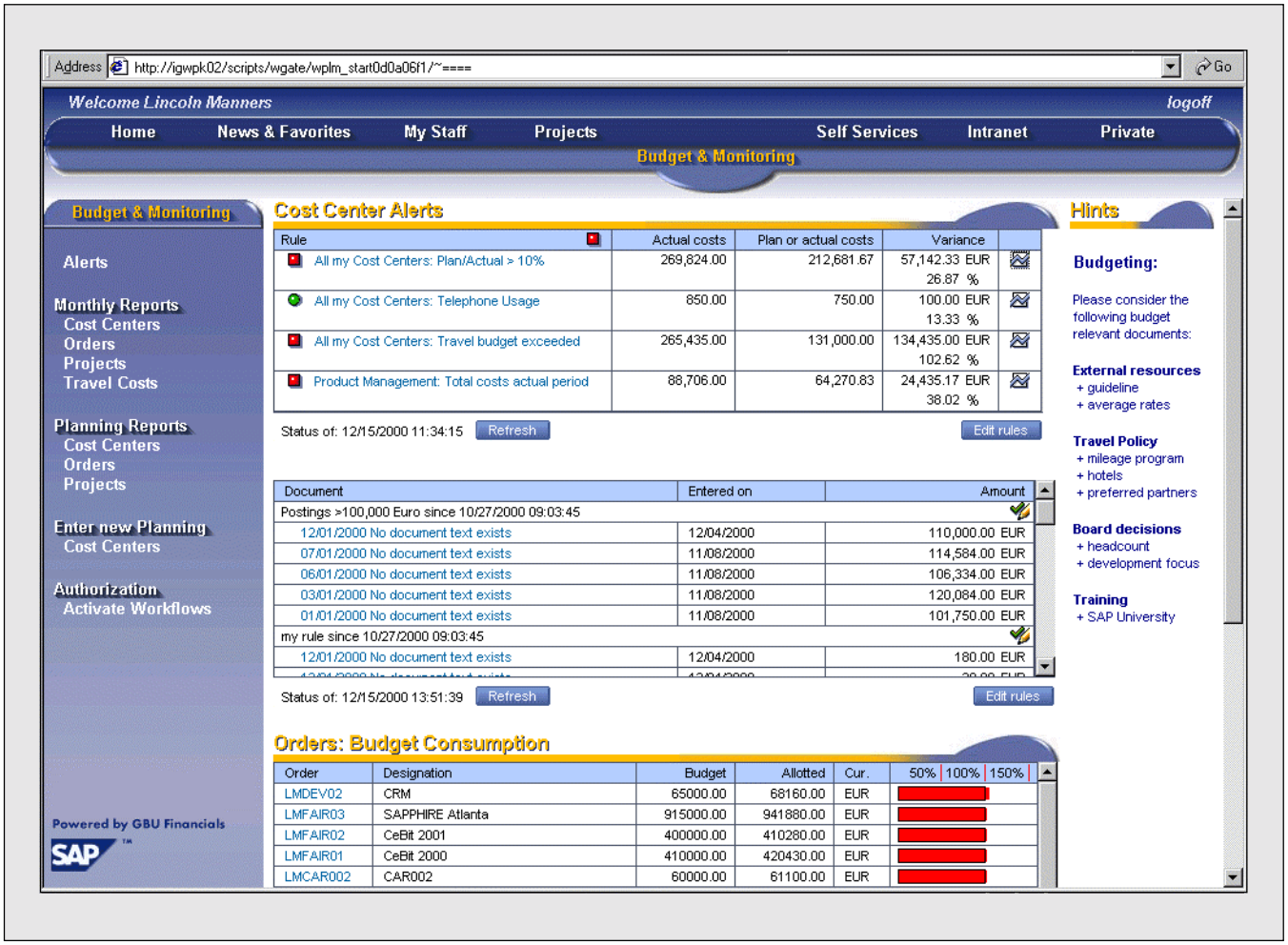
MiniApps are "smart," stateless, role-specific Web applications that are fully integrated in the mySAP Workplace. Take a look at **Figure 1**, which shows an example of the home page that is presented to a user whose mySAP Workplace has been configured with the "Line Manager" role. In the "LaunchPad" on the left, the user can click on the application and have it launched in the "WorkSpace" pane on the right. In this screen shot, you can see the "Cost Center Alerts" and the "Orders: Budget Consumption" MiniApps, which are assigned to the Line Manager role.

SAP delivers a variety of MiniApps such as these, which offer access to various SAP component systems. SAP also provides ample opportunity for developers to create their own MiniApps and integrate them with the Workplace. You can use HTML, DHTM, JavaScript, Active Server Pages, and Java to do this. And, as of Release 4.6C, you can now create MiniApps with the ABAP Workbench.

With Basis Release 4.6C, SAP introduced the Web Application Builder, a new platform for developing ITS-based[1] Web applications

[1] The ITS (Internet Transaction Server) is an SAP product that acts as a gateway between the Web server and the SAP application server (for Release 4.0B and up).

Figure 1 mySAP Workplace Home Page Configured with the "Line Manager" Role



within the comfort of the ABAP Workbench environment.[2]

MiniApps do not have to be ITS-based, nor do they have to be built with the Web Application Builder, so why do I favor this approach and this new development tool? What do you need to know about the architecture of the Workplace before you can attempt to build an ITS-based MiniApp? What development objects will you need to implement? What utilities are available to help you develop those objects? And what steps must you take in order to build that MiniApp from start to finish? These are the questions I will answer in this article.

Why Build ITS-Based MiniApps?

The Web Application Builder supports both the "inside-out" and "outside-in" programming models:

- **Inside-out** applications are based on existing R/3 transactions, which can be called from the Web browser using the HTTP protocol. The

[2] The Web Application Builder is an alternative to the PC-based development tool SAP@Web Studio. This new development tool allows you to develop ITS-based applications *outside* the SAP system.

transaction is completely executed within the SAP system and the application state is maintained within the SAP application server.

- **Outside-in** applications use BAPIs and function modules for business logic. The presentation logic runs completely on the ITS and is implemented using HTML^{Business} and flow logic (see sidebar). This is the programming model we will employ when building ITS-based MiniApps. An ITS-based MiniApp uses the flow logic and runs on the ITS.

As an ABAP developer, I favor the outside-in development approach for three reasons:

- ITS MiniApps support reuse very well. All appropriate function modules and BAPIs from Release 4.0B onward can be used for this kind of MiniApp.

- The Web Application Builder is a tool designed specifically for developing ITS-based Web applications, and yet it is fully integrated into the ABAP Workbench, which means it confers all the benefits of the ABAP Workbench, including:

  ✓ Connections to the Transport Organizer

  ✓ Use of the standard Workbench navigation functions

  ✓ The ability to administer language resources

  ✓ Version management

- This approach does not require an ABAP developer to surmount a large learning curve. The ABAP Workbench is a familiar environment, so mastering the Web Application Builder will not take long at all. That should buy you extra time to focus on your application's visual and interactive design (the importance of which, in this day and age, cannot be overstated) and on implementing function modules and BAPIs within the context of a Web application.

HTML^{Business} and Flow Logic: The Keys to ITS Programming

HTML^{Business} is an SAP-specific macro language designed specifically for the ITS. When building MiniApps, HTML^{Business} allows you to merge data from the SAP system into HTML templates dynamically. HTML^{Business} contains keywords, expressions, loop and condition statements, and many predefined layout functions. To use the HTML^{Business} Library, simply add an include statement at the start of an HTML template.

Flow logic defines the dialog flow of Web applications (such as ITS-based MiniApps) that use the ITS as their runtime environment. Flow logic uses a small subset of XML (eXtensible Markup Language) elements to make the dialog logic relatively simple. A typical flow consists of a set of states and events. States generally contain module calls (BAPI or RFC calls) for communicating with the backend, while events define the transitions possible between states and trigger the processing of the state. You can create the flow logic for each HTML template in the ABAP Workbench's Web Application Builder, and then edit it using the Flow Editor or Flow Builder.

The mySAP Workplace Architecture — What You Need to Know

ITS-based MiniApps do not run as standalone applications. They are always integrated in the Workplace infrastructure as a part of a role definition. It is therefore critical that you understand how these elements work together.

Figure 2 The Three Layers of the mySAP Workplace Architecture

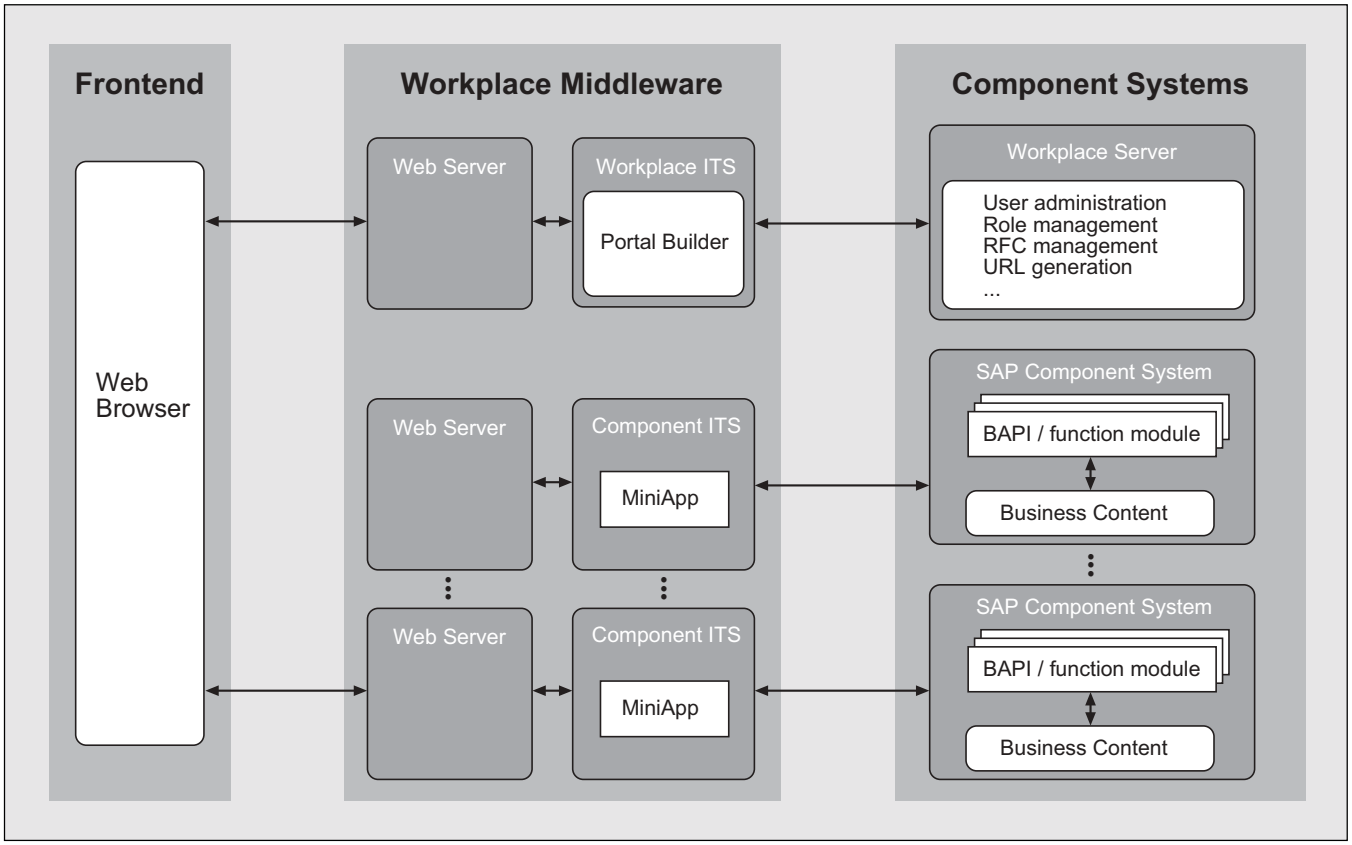


Figure 2 displays the three layers of the mySAP Workplace architecture:

- **Frontend:** To display the mySAP Workplace, users need a Web browser (Internet Explorer 4.0 or 5.0) on their frontend. The browser uses HTTP to communicate with the Web server (part of the Workplace Middleware layer).

- **Workplace Middleware:** The Workplace Middleware links the frontend to the target component system. Between each MiniApp and the target component it is designed to access, there is one Web server and one instance of an ITS server. So what is that "Portal Builder" you see in Figure 2? This is a special ITS instance that communicates directly with the Workplace server. The Portal Builder is used to generate a specific mySAP Workplace for each user. The Portal Builder gets information about the current user's roles from the Workplace server and creates the Workplace structure (LaunchPad and WorkSpace).

- **Component Systems:** The Workplace server is an SAP Basis system providing special mySAP Workplace functions, like central user administration, role management, RFC management, and URL generation. SAP component systems or non-SAP systems (not shown in Figure 2) are connected to the Workplace server by RFC (Remote Function Call) connections. Any SAP system from Release 4.0B onward can be considered a valid SAP component system. The component systems actually trigger the MiniApps. They call BAPIs or function modules, then pass the resulting output data to the appropriate instance of the ITS server. The formatted HTML page is then passed to the browser via the associated Web server.

Steps and Tools for Developing an ITS-Based MiniApp

There are five steps to building an ITS-based MiniApp:

1. **Pick the appropriate BAPI or function module.** In this step, you specify the underlying business logic and data collection mechanism for your MiniApp. ITS-based MiniApps perform data retrieval by calling a function module or a BAPI, so you need to identify the appropriate one(s). In most cases, you can use an existing function module or BAPI. MiniApps will support any function module or BAPI from Release 4.0B onward.[3] For example, in the next section, where we build a MiniApp to perform currency conversion, we will utilize the `GetList` BAPI for the `Currency` object.

2. **Design the interface.** In this step, you specify the look-and-feel of your MiniApp's user interface. The interfaces and functions of a MiniApp are much less complicated than those of a full-fledged application. Take advantage of this. A typical interaction between the user and the system takes place on just a single screen. Make your design simple, uniform, and task-oriented.[4]

3. **Implement the Internet service.** This step constitutes the bulk of the development effort. At the heart of every ITS-based MiniApp is an Internet service. A typical Internet service includes:
   - A service definition
   - HTML templates
   - Language resources
   - MIME objects (where applicable)
   - A theme

[3] You can implement the development of cross-release MiniApps using an encapsulation module, which you create in a development system (Release 4.6C or higher). This encapsulation module acts as a cross-release interface between the development system and the different releases of component systems (4.0B, 4.5B, and 4.6B) in which MiniApps will access data.

[4] Valuable tips on visual and interactive features, along with many examples and ideas for your MiniApps, can be found at the MiniApp Community home page (**www.sap.com/MiniApps**).

Figure 3 provides an overview of these elements.

Figure 3 Elements of an Internet Service

Service definition

The service definition contains parameters that describe the type of the service and how it is to be executed on the ITS.

HTML templates

HTML templates form the user interface for an Internet service. They contain the HTML tags and formatting information used to display R/3 data to the Internet customer. The templates are built using presentation logic (HTML^{Business}) for the user interface, and flow logic for looping through the R/3 data to be filled into the template.

Language resources

Language resources enable you to make your application multilingual (e.g., when users start the service, the text appears in their native language). The Internet service's language-specific elements are defined as "variables" in the service's template. The service then has a language resource defined for each available language, in which the value of each variable in the template is defined (e.g., English, German, etc.).

MIME objects

MIME objects enable you to identify the icons, graphics, Java applets, and sound or video components to be inserted into the service to enhance the user interface for the Web environment.

Theme

The theme specifies the appearance of a user interface. Each theme has its own set of HTML templates, language resources, and MIME objects, and gives a user interface a particular appearance. Customers can use the delivered default theme (99), or they can copy it and then make modifications. For instance, you can exchange icons, buttons, or colors. (An Internet service can contain several themes.)

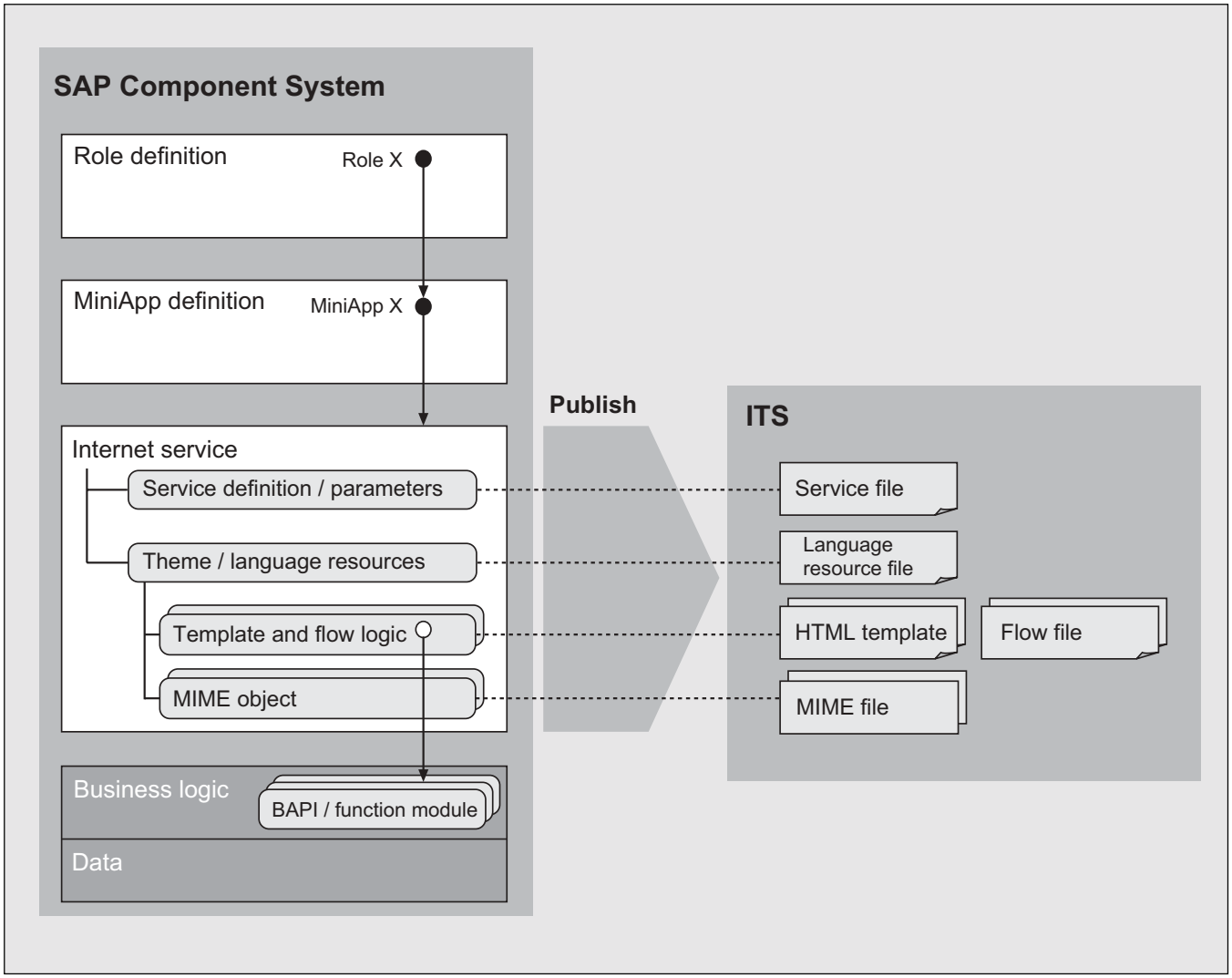
Figure 4 Tools for Developing an ITS-Based MiniApp

HTML Editor	Used to edit the source code in the HTML templates for Web applications. Source code generally includes HTML elements, keywords, expressions, statements, and HTML^{Business} functions. You can also define your own JavaScript functions.
Patterns and Wizards	To ensure that the source code you are editing is free of errors, you may use patterns and wizards for HTML^{Business} functions. Patterns allow you to insert source code for HTML^{Business} functions to an HTML template more easily. You can use wizards to create more complex elements, such as tabs, group boxes, or buttons that trigger events.
Flow Editor	Used to implement the interaction design of an HTML template using flow logic statements. You have to edit the necessary language elements of the flow logic manually. When using the Flow Editor, you must be familiar with the exact syntax of the flow logic. The Flow Editor does not carry out a syntax check.
Flow Builder	Used to create flow logic step by step, even if you don't know the exact syntax. The correct flow syntax is generated automatically. The Flow Builder supports you by providing the input help and navigation functions you need. You can also use the check function to check the dialog logic for inner consistency. Using the check function, you can guarantee that the flow logic is internally consistent and semantically correct.
MiniApp Maintenance	In Basis Release 4.6D,[5] MiniApp maintenance will be incorporated in the Web Application Builder, so that all the functions needed to create and edit MiniApps are available in one tool, including: • Assigning the MiniApp definition to an Internet service or to a URL • Editing the attributes the system needs to connect the application to the MiniApp framework • Defining parameters for transferring values between the MiniApp definition and the service assigned to it

[5] Basis Release 4.6C already contains the basic functions you need to create and edit MiniApps. However, in 4.6C, you have to perform these functions directly on the relevant table; they are not available in one maintenance tool.

Your job is to build the HTML templates, specify the parameters of the service definition, specify a theme, and identify which textual elements need to be earmarked as "language resources" so that they can be isolated from the logic of your application and dealt with in a way that facilitates multilingual support of your MiniApp. The Web Application Builder provides you with the tools you need to do all this (see **Figure 4**), and, as you will soon see when we build our Currency Converter MiniApp, prompts and guides you every step of the way.

Figure 5 Relationships Between Each SAP Component System and the ITS

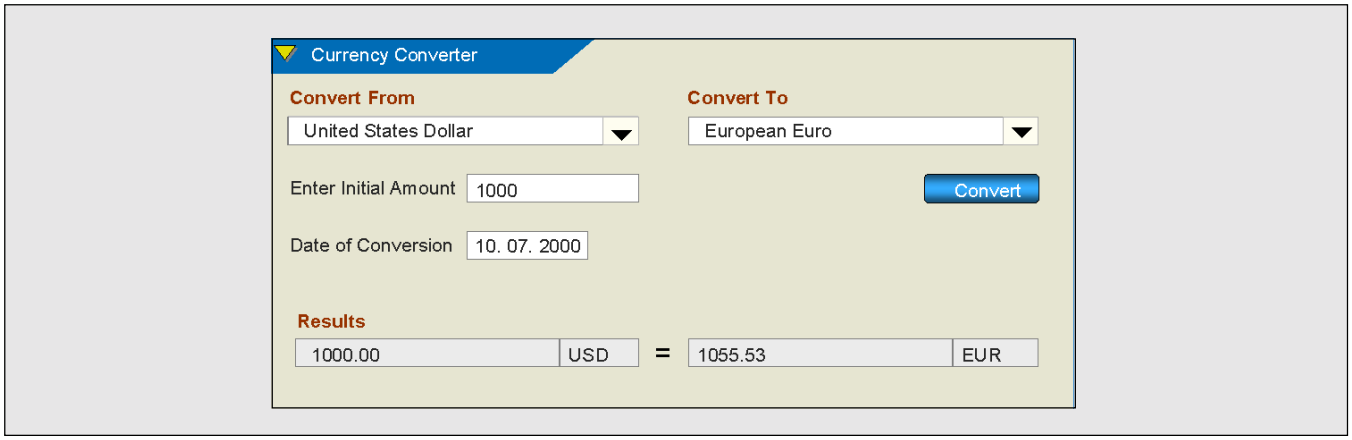


4. **Publish the Internet service on the ITS.** After you have developed an Internet service in the Web Application Builder, you publish it on the ITS. Publishing is the process by which you transfer, or copy, all the components of your Internet service from the SAP Repository to the ITS file system.

5. **Integrate the MiniApp in the Workplace.** The last step is to create a *MiniApp definition* development object in the Web Application Builder. This object contains the administrative information that the system needs to be able to integrate an Internet service in the Workplace and to use the generic functions of the MiniApp framework. This MiniApp definition can be assigned to one Internet service, or to any URL, and is used as part of a role definition.

Figure 5 shows how all these objects work together, and how the development objects in the Web Application Builder are represented on the ITS after they are published.

Figure 6 Our MiniApp's User Interface



Building a Simple "Currency Converter" MiniApp

So much for the theory. Let's dive into an example and build a MiniApp for converting local currencies to a variety of foreign currencies — in some cases, using a rate from a previous date rather than today's date. All the relevant conversion data is stored in tables in an SAP database.[6]

Step 1: Implementing Business Logic and Data Collection

First, the MiniApp needs a complete list of the relevant currencies, which can be obtained by calling the standard BAPI `GetList` from the `Currency` business object. The `CURRENCY_LIST` output table contains the fields with the currency codes and their associated long texts that we can use in our template.

We will be using a function module of our own design, `CURRENCY_CONVERSION_MINIAP`, implemented specifically for this MiniApp, since at present there is no BAPI for currency conversion.

[6] Don't confuse this MiniApp with the "Currency Converter" that ships with the mySAP Workplace. The MiniApp I am describing here is quite different! I've optimized this for teaching purposes, not for actual currency conversion.

The system passes values for each currency unit along with the conversion date to this function module. The function module returns the converted rate for the foreign currency.

Step 2: Specifying the User Interface

There are several ways to design the user interface, even for such a simple application. **Figure 6** shows my design of a template for the MiniApp. The available currencies are displayed in two dropdown boxes. One input field prompts the user for the initial amount to be converted. The other allows the user to enter a conversion date. This date field is optional; the current system date is fetched by default. When users trigger the conversion by clicking on the "Convert" button (which should be in a contrasting color or highlighted in some other way), the system returns the two amounts in the appropriate two "Results" fields. There is no screen change at any point.

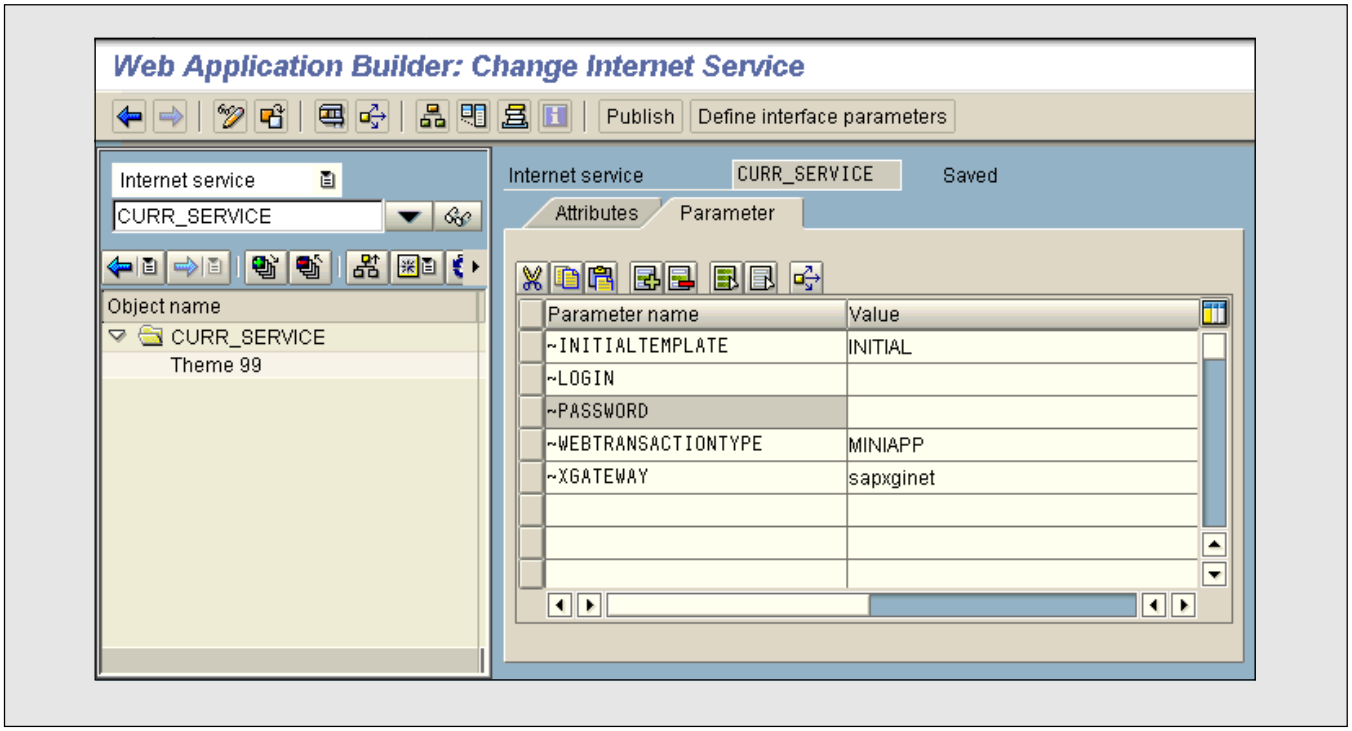
Step 3: Implementing the Internet Service

To implement the Internet service for this MiniApp, we proceed as follows:

1. Start the Object Navigator (transaction SE80).

2. From the object list, choose "Internet service" and enter the name for the service we want to

Figure 7 Creating an Internet Service for Our Sample MiniApp



create. We will enter "CURR_SERVICE" — the technical name for our example Internet service.

3. We confirm our entry.

4. In the dialog box that appears next, we choose "Web application" and enter the name of the first template, "INITIAL". This template will be used as the initial template to start the service on the ITS.

5. We have now created the service. Save it and assign a development class to the service.

The service we have created is displayed in the object list tree structure shown in **Figure 7**. By default, "Theme 99" is automatically assigned to the service. In addition, three of the service definition parameters, along with their values, were automatically generated for the service:

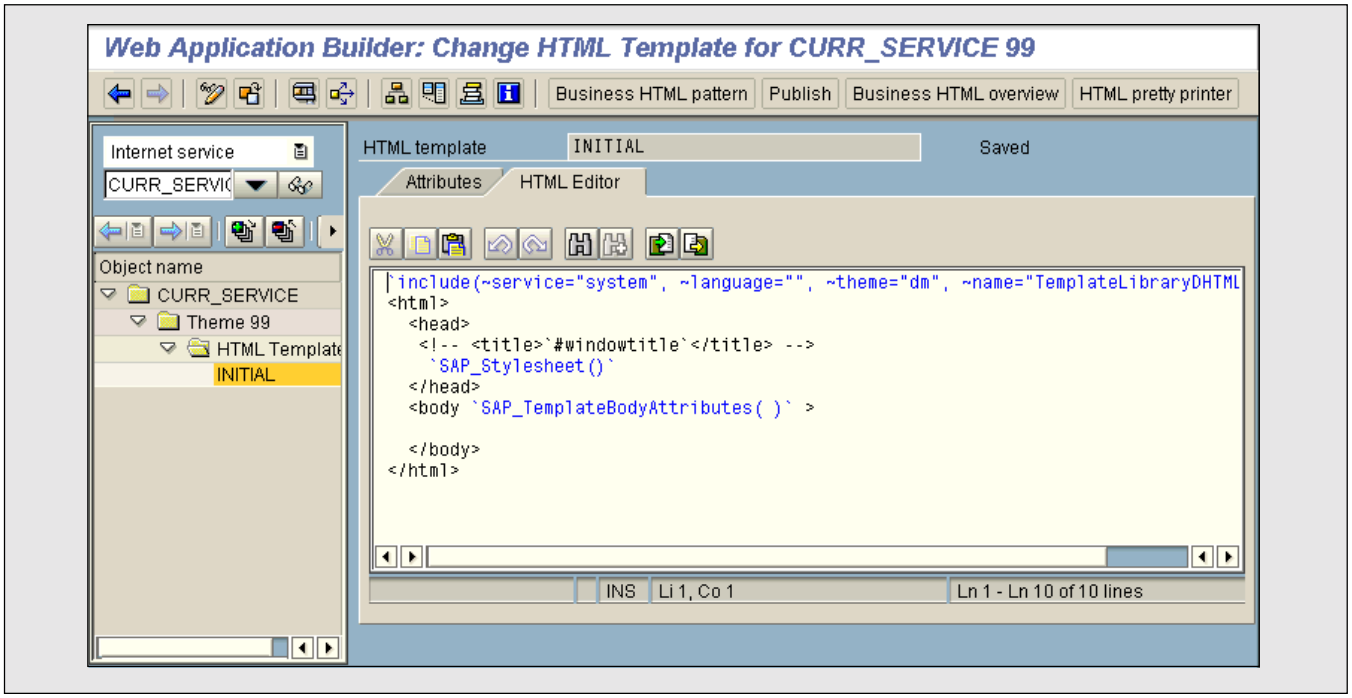
- **~INITIALTEMPLATE**, which contains the name of the HTML template that is used to start the MiniApp.
- **~WEBTRANSACTIONTYPE**, which defines the type of the created service.
- **~XGATEWAY**, which contains the name of the X gateway for communication with the ITS.

These parameters specify the type of Internet service and the way the ITS will execute it.

What we have just done is create a service definition in the Web Appli-cation Builder. "CURR_SERVICE" is a Web application of type "MINIAPP," which is started with the "INITIAL" template using the X gateway "sapxginet."

Now we are ready to create an HTML template. We select "CURR_SERVICE," the service we have

Figure 8 *Creating the HTML Template*



✓ *Tip*

When you create an Internet service for a MiniApp, it is recommended that you add the service parameters ~LOGIN and ~PASSWORD without values, so that later, when the MiniApp is integrated into the Workplace, the system can get the user data from the single sign-on context (cookie, digital certificate, etc.). When users log on to the mySAP Workplace, their user information is stored in that context; otherwise SSO could not work for the MiniApp.

just created, from the object list tree and choose **Create** → **Template** from the context menu. In the "Create template" dialog box, we have the choice to assign a new theme or keep the default theme (Theme 99) that was automatically assigned. We keep the default theme, and enter a name for the first HTML template (INITIAL) in the appropriate fields. This name matches the value of the ~INITIALTEMPLATE parameter shown in Figure 7.

We then save our entries and assign a development class to the template.

The Web Application Builder inserts the template we have created into the object list tree structure shown in **Figure 8**. The pre-generated HTML and HTML^{Business} source text is displayed in the HTML Editor.

This initial source text contains, among other things, the include statement that adds the HTML^{Business} standard library to the template. HTML^{Business} elements are enclosed in inverted apostrophes (‘…‘), and are highlighted in blue in the editor to distinguish them from standard HTML.

The HTML^{Business} Library includes a large number of functions for displaying screen elements, all of which comply with the design criteria specified

Listing 1: Initial Structure of the HTML Source Code for the Currency Converter

```
<body 'SAP_TemplateBodyAttributes( )'>

  <table>
    <colgroup>
        <col width=170>
        <col width=40>
        <col width=10>
        <col width=170>
        <col width=40>
    </colgroup>
    <tr>
        <!--  Label fields for Currency                -->
    </tr>
    <tr>
        <!--  Drop down list box for "From" currency    -->

        <!--  Drop down list box for "To" currency      -->
    </tr>
    <tr>
        <!--  Input field for initial amount           -->

        <!--  Button to perform calculation            -->
     </tr>
     <tr>
        <!--  Output/Input field for Date of conversion -->
     </tr>
     <tr>
        <!--  Output fields for amount                 -->
     </tr>
  </table>
</body>
```

by SAP. If we start with the basic structure of the HTML source code, it is easy to add these functions to our template as patterns and then fill them with the values we need. For more complex elements — like creating a button that triggers an event — it is better to use a wizard.

For the Currency Converter, we will use HTML^{Business} statements and expressions to reference fields from the SAP system, and create loops and conditional statements for these fields in the HTML template.

We create a basic HTML structure using an HTML table and specify some of the formatting entries at this point (see **Listing 1**[7]). For listings of

[7] Here, and in the following listings (which are excerpted from the code in the appendix), the HTML^{Business} elements are enclosed in inverted apostrophes (' … ') and highlighted in bold to distinguish them from standard HTML (normally they are highlighted in blue).

the presentation source code, the flow logic code, the service parameters, and the theme parameters of the Currency Converter MiniApp, refer to the appendix that accompanies this article. You can also find this information available for download at **www.SAPpro.com**.

The best way to add functions for screen elements like input fields, output fields, labels, and buttons is by inserting a pattern in the source code.

To do this, we:

1. Set the cursor in the HTML Editor at the appropriate point and choose the "Business HTML pattern" button, shown in Figure 8.

2. Select a pattern function and double-click this function to display the description, parameters, and documentation associated with it.

3. Confirm our choice by clicking on the "Insert " button.

The source code of the HTML^{Business} function we have chosen will be inserted at the cursor position in the HTML Editor. All optional parameters for that function are initially commented out. We can then fill these parameters with values as appropriate.

Listing 2 shows how you would define the input field for the currency. The value that the user enters, `LocalCurrencyAmount`, sets the corresponding importing parameter for the currency conversion function module. To specify a label, we use the language-independent placeholder, `#amount`, which we will later include as a language resource. The other parameter values specify the width of the input field on the screen.

A button that triggers a flow event requires other functions along with the display function. The best way to link these functions to the button is to use a wizard.

We start the wizard by clicking the "Business HTML pattern" button. Choose the wizard for buttons and then drag&drop it into the HTML template. The wizard guides us through the complete process until the element is ready for use.

We make the required entries and then choose the "Complete" button to generate the source code. The Web Application Builder inserts the necessary elements into your HTML template: the source code for the HTML^{Business} function for the pushbutton; the JavaScript function for coupling a flow event; and the HTML tags defining a form (**Listing 3**). The system automatically assigns the `action` variable to the `wgateURL()` function and dynamically generates the URL for the current Web server for that system.

When we create a dropdown box, we need to insert functions not only for displaying it on screen, but also for filling it with data from the appropriate SAP system table. In our example, this data is passed to the template using the array `CURRENCY_LIST-CURRENCY[i]` after the `Currency.GetList` BAPI has been called. The long text `CURRENCY_LIST-LONG_TEXT` for each available currency unit is then set as the content of the dropdown box. Once the user chooses an entry from `LocalCurrencyUnit`, the system sets the corresponding importing parameter of the currency conversion function module.

Listing 4 shows how easy it is to reference fields from the SAP system. The syntax of the `FOR` and `IF` statements will be familiar to many programmers, since it is similar to that in common languages like C and JavaScript.

To make the HTML template for our MiniApp multilingual, we create language resources. To do this, we only need to make sure that all the language-specific texts in the template are replaced by placeholders — for example, `'#amount'`. We will then assign the texts we want in the original language to these placeholders.

Listing 2: Defining the Currency Converter Input Field

```
<!--  Input field for initial amount                -->
'SAP_TemplateEditableField(align="left"
                           ,fieldLabel="#amount"
                           ,fieldLabelWidth="90"
                           ,"Amount_ID"
                           <!--,inspectionText=""-->
                           ,maxlength=30
                           ,name="LocalCurrencyAmount"
                           <!--,onchange=""-->
                           <!--,required=""-->
                           ,size=15
                           <!--,type="SAP_WEBGUI"-->
                           ,value= LocalCurrencyAmount
                           <!--,width=""-->)'
```

Listing 3: Elements Added to the HTML Template

```
<script language="JavaScript">
  function convert()
  {
    document.input_form.elements['~event'].value = 'onConvert';
    document.input_form.submit();
  }

</script>

<form name="input_form" action="'wgateURL()'"  method="post">
<input type="hidden" name="~event" value="">

'SAP_TemplateLargeActionButton("CalcButton_ID"
                               ,buttonLabel=#convert
                               ,onclick="convert()")'
```

Listing 4: Referencing Fields from the SAP System

```
<!--   Drop down listbox for "From" currency     -->
 'SelectionIndex1 = 1'
'for ( i = 1 ; i <= CURRENCY_LIST-CURRENCY.dim ; i++ )'
    'if ( LocalCurrencyUnit == CURRENCY_LIST-CURRENCY[i] )'
        'SelectionIndex1 = i'
        'i = CURRENCY_LIST-CURRENCY.dim'
    'end'
'end'
```

(continued on next page)

(continued from previous page)

```
'SAP_TemplatePulldownField(<content="CURRENCY_LIST-LONG_TEXT"
                           <!--,contentdim=""-->
                           <!--,fieldLabel=""-->
                           <!--,fieldLabelWidth=""-->
                           ,"FromCurr_ID"
                           <!--,inspectionText=""-->
                           ,key="CURRENCY_LIST-CURRENCY"
                           ,name="LocalCurrencyUnit"
                           <!--,onchange=""-->
                           ,selIndex=SelectionIndex1
                           <!--,selKey=""-->
                           <!--,size=""-->
                           <!--,type="SAP_WEBGUI"-->
                           ,width="220")'
```

✓ *Tip*

In general, there are three possible ways of creating language-independent HTML templates for MiniApps:

- *Use language information already available in the SAP system. This is the simplest approach, but it does not necessarily provide texts for all the elements in the template.*
- *Use language-specific templates: write a separate set of templates for each supported language, and assign each set to a separate theme. The disadvantage of this approach is that any changes made to the templates have to be made several times.*
- *Use language resources for all the templates in a service. We can then edit the language-specific texts separately from the template itself. We establish a reference in the template using placeholders. We then create texts for the original language, which appear in translation worklists. (This third option is the one we are using in our example.)*

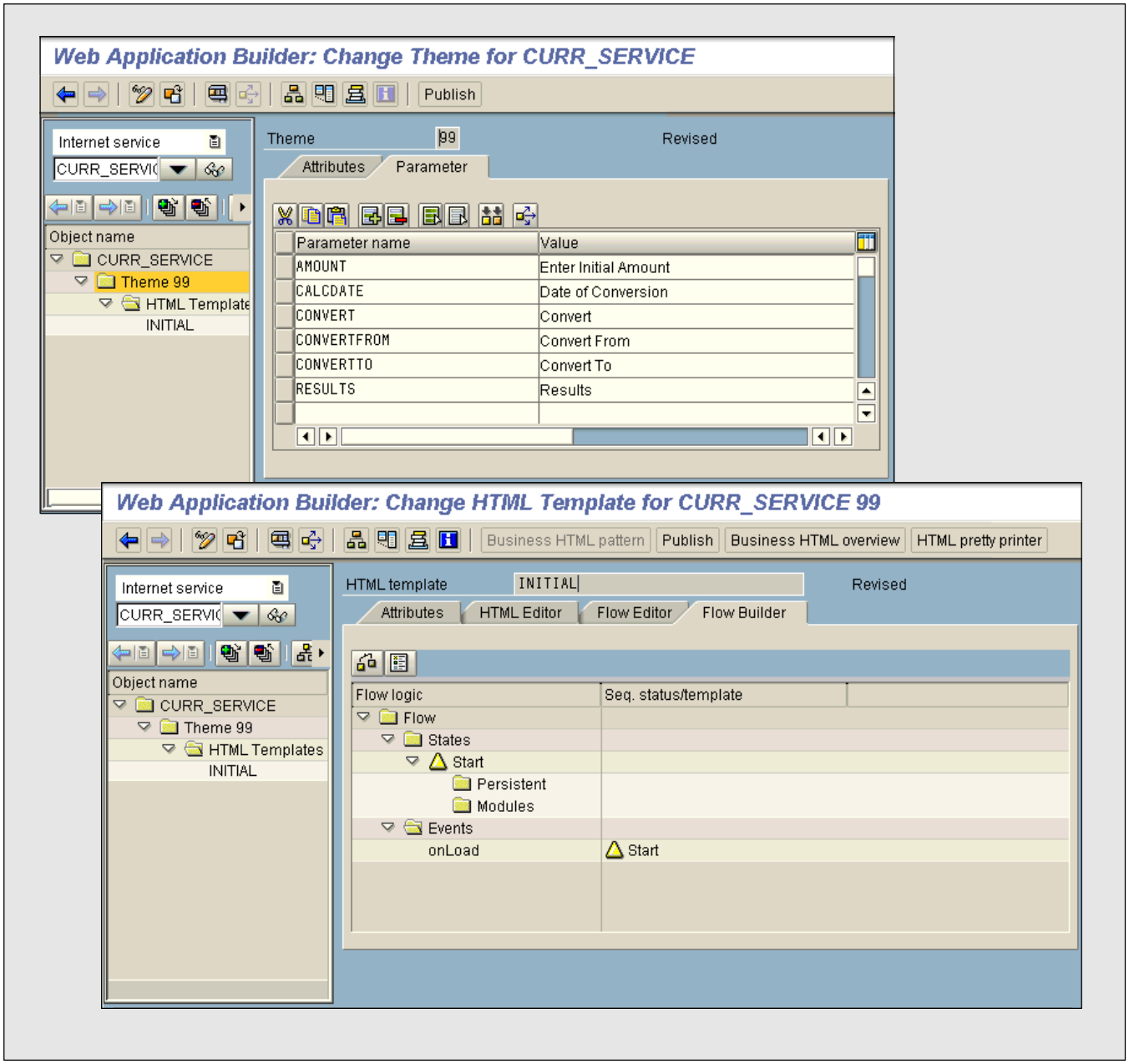
To enter language-specific texts, we double-click a placeholder in the template. Since language resources are created as theme parameters, the system automatically displays the associated theme. We then choose the "Compare parameters" function, which compares the placeholders in the template with the list of theme parameters and adds any new ones to the list. Finally we enter the language-specific texts in the original language as values for each parameter. (If you look at the first screen in **Figure 9**, you will see that I've entered values for each parameter in English.)

The new theme parameters are now part of the service. They are translation-relevant parts of the R/3 Repository object, and as such will enter the translation workflow when you release the service. At runtime, the ITS recognizes the placeholders and replaces them with texts in the appropriate language.

To implement an interactive design, we need to define the flow logic, which specifies:

- How the system reacts to user events
- When the system triggers module calls (BAPIs or function modules)

Figure 9 Specifying Theme Parameters in the HTML Template and Starting the Flow Builder

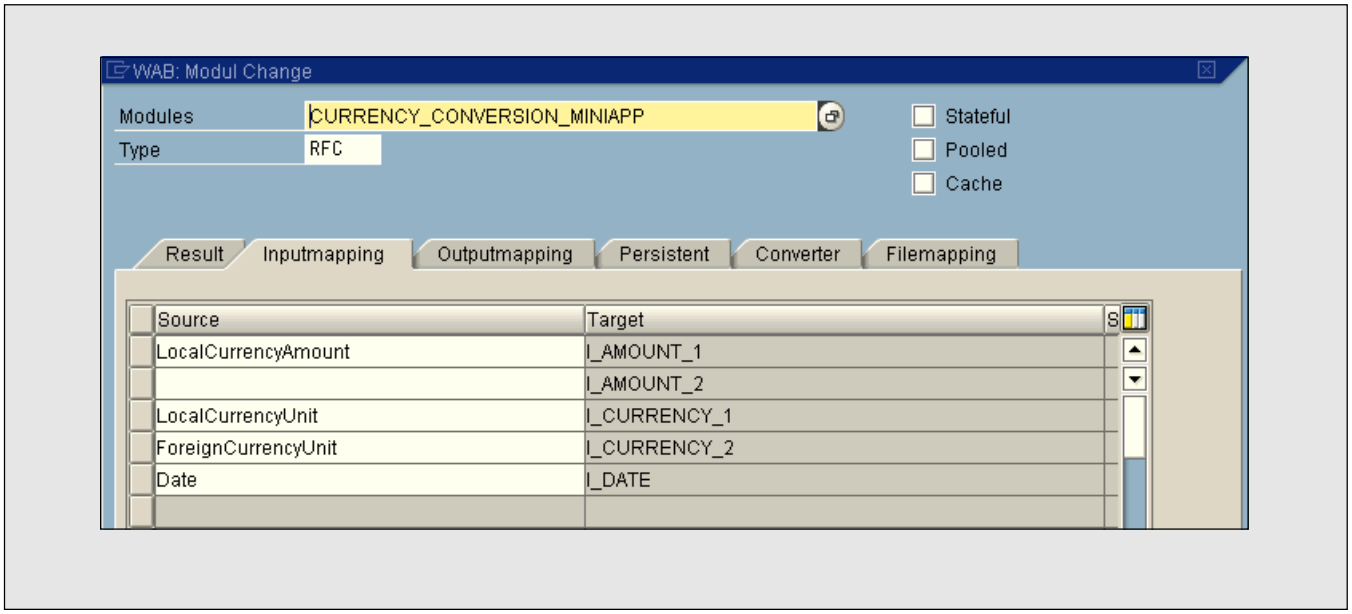


- How the system transfers data between the template and the SAP system

We want to take advantage of the many benefits of the Flow Builder and create a dialog logic by defining the appropriate events, along with their associated module calls.

To start the Flow Builder, choose the menu path **Edit → Create flow logic** and select the "Flow Builder" tab. For each new HTML template, the Flow Builder provides an initial representation of the flow logic with the initial state, "Start," and the initial event, "onLoad" (see the second screen in Figure 9). This event will be processed when

Figure 10 Mapping Variable Names to Importing Parameters



the template is displayed for the first time. The event "onLoad" is associated with the "Start" state. We will now see how to define module calls in a state.

In the initial tree structure, double-click the "Start" state for the first module definition. We enter the name of the BAPI or the corresponding RFC function module, using the possible entries help (F4). We double-click a module name and enter the mapping parameters (and other attributes if necessary). The RFC parameters are already shown, either as "Target" (for mapping the input of the importing parameters) or "Source" (for mapping the output of the exporting parameters).[8]

To create a new event, double-click the initial event "onLoad" or choose the appropriate function from the context menu.

[8] Note that there is no standard procedure available for handling the table in its entirety between the system and the flow logic. You need to export or import the single parameters of a table ("TableName-parameter X," "TableName-parameter Y," etc.).

✓ Tip

Figure 10 *shows how we map variable names in the template to the appropriate importing parameters in the conversion module. This mapping is not strictly necessary, but it does make the source code more readable by allowing developers to assign meaningful names to variables.*

Enter the name of the state in the new line, then the name of the next state; confirm your entries. We will now look at the result (**Listing 5**). We could also display this result by switching to the Flow Editor. However, the Flow Builder applies the correct flow syntax to our entries. Now, let's take a closer look at the dialog logic.

When the template is loaded, the `onLoad` event (linked to the `Start` state) is triggered first. Then the system calls the `Currency.GetList` BAPI,

Listing 5: Creating a New Event

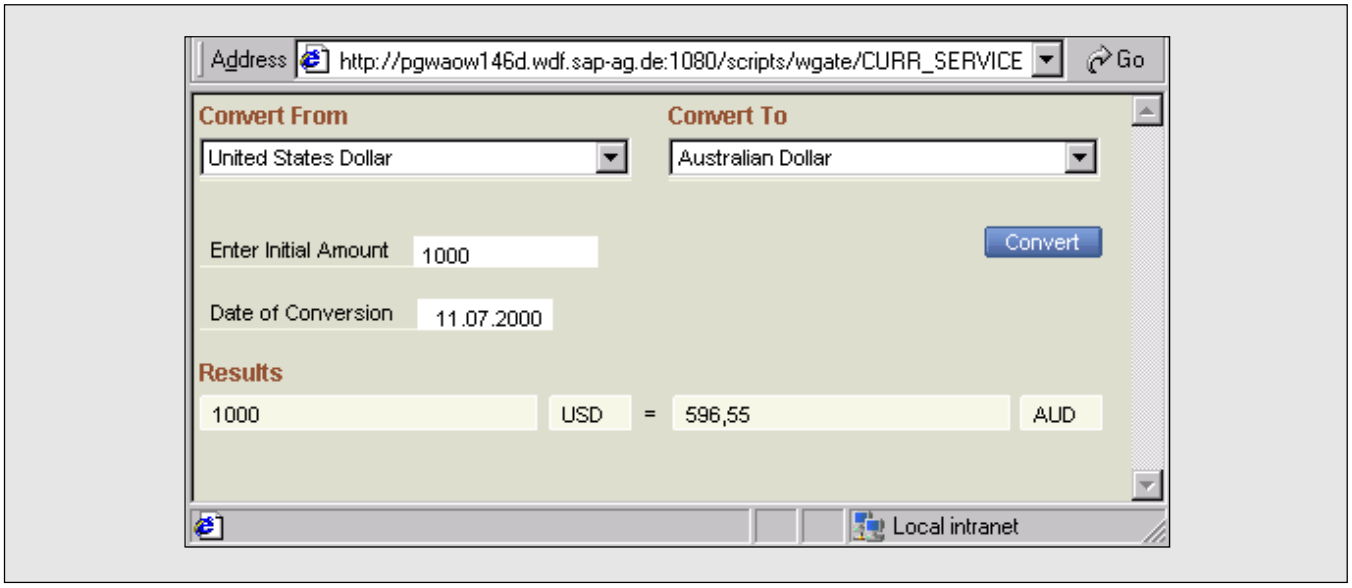
```
<flow>

 <state name="Start">
      <module name="Currency.GetList" type="BAPI">

         <persistent name="CURRENCY_LIST-CURRENCY"/>
         <persistent name="CURRENCY_LIST-LONG_TEXT"/>

      </module>
 </state>

 <state name="convert">
      <module name="CURRENCY_CONVERSION_MINIAPP" type="RFC">

           <inputmapping source="LocalCurrencyAmount" target="I_AMOUNT_1"/>
           <inputmapping source="LocalCurrencyUnit" target="I_CURRENCY_1"/>
           <inputmapping source="ForeignCurrencyUnit" target="I_CURRENCY_2"/>
           <inputmapping source="Date" target="I_DATE"/>

           <outputmapping source="E_AMOUNT_2" target="ForeignCurrencyAmount"/>
           <outputmapping source="E_DATE" target="InitialDate"/>

      </module>
 </state>

 <event name="onLoad" next_state="Start"/>
 <event name="onConvert" next_state="convert"/>

</flow>
```

which reads all the available currencies from the `CURRENCY_LIST`. The fields for the currency codes and their associated long texts are set to `Persistent`, since they should be available for the entire ITS session.

The user triggers the `onConvert` event by clicking a button, which launches the currency conversion procedure in the assigned state, `convert`. All the variables in the template that correspond to importing parameters in the function module are mapped to table fields (input mapping). Output mapping takes place for all the variables corresponding to exporting parameters.

If you are wondering what happens if multiple texts are the same for different currencies , the answer is that the currency is always identified using the currency code and there is a 1:1 relationship between the currency code and text.

Figure 11 Execute and Test the Service As a Standalone Application



Step 4: Publishing and Testing the Service

For an Internet service to be executed by the ITS, it must be stored in the ITS file system, a process known as *publishing* the service. We can choose to publish the entire service or just parts of it. When we publish the whole service, the corresponding Internet service and its HTML template are placed in the file system of the ITS server.

To publish the service, we select the service in the object list and choose **Publish → Complete service** from the context menu.

After we have published the whole service, we can start and test it. For doing this, we choose F8. The system starts the service, using the following HTTP address:

http://<web_server><web_path_prefix>/<service>/!

and displays the HTML page in the browser.

Now we execute and test the service for the MiniApp as a standalone application (**Figure 11**).

Step 5: Integrating the Application in the Workplace

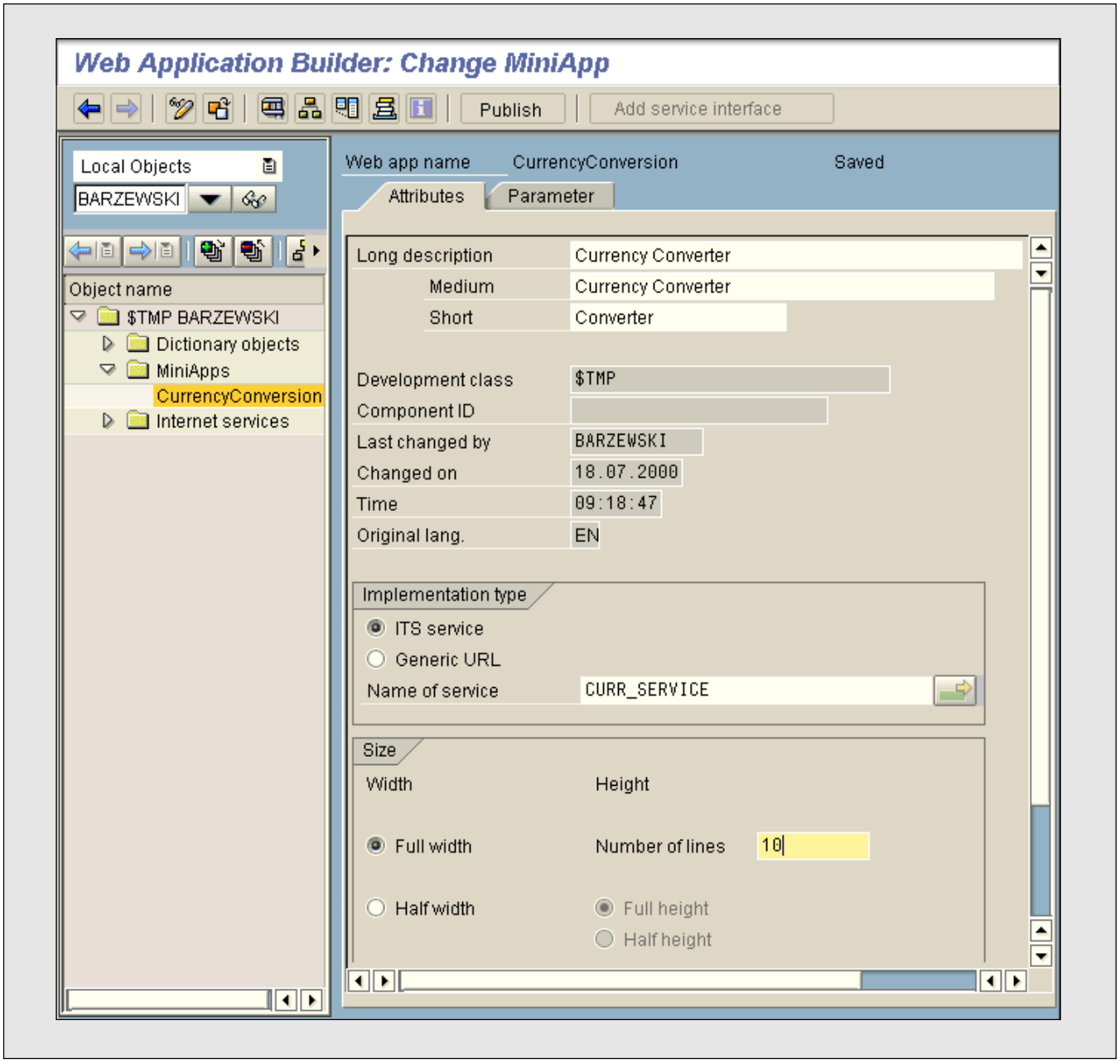
Until now, we have dealt with our MiniApp service without any reference to the mySAP Workplace. To integrate the service in the Workplace, however, the system needs certain administrative data — for example, how to display the MiniApp and what generic Workplace functions it should use.

To create the MiniApp definition[9] using "MiniApp maintenance":

1. We call MiniApp maintenance from the Object Navigator by clicking the development class and choosing **Create → MiniApp** from the context menu.
2. In the "Create MiniApp" dialog box, we enter the name and a description for our MiniApp and confirm our entries. The Web Application Builder displays the "Attributes" of the MiniApp (see **Figure 12**).

[9] In Basis Release 4.6C, we created this definition directly in the appropriate tables. In Release 4.6D, however, we do this centrally in "MiniApp maintenance." For more information on this procedure in Basis Release 4.6C, see the mySAP Workplace documentation.

Figure 12 Attributes of the MiniApp in mySAP Workplace



3. We enter a medium and short description for the MiniApp. The description is used in the Workplace as the frame title for the MiniApp. After we have included the MiniApp in the Workplace, one of these descriptions will be used as the tray header of the MiniApp. The description used depends on the dimensions of the MiniApp in the Workplace frame.

4. We accept the default option "ITS service" as the "Implementation type" and enter the name of the service from which the MiniApp is to be called.

5. We specify the size the MiniApp will have within the Workplace frame and save our entries.

   As shown in Figure 12, we have created a

MiniApp definition "CurrencyConversion" and assigned it to the service "CURR_SERVICE."

> **✓ Tip**
>
> *If you specify the size of the MiniApp, make sure that all its contents fit the Workplace frame. The dimensions of the MiniApp do not change dynamically when the data is changed. If the content is too big for the size defined, scroll bars appear. Wherever possible, however, you should avoid having scroll bars in MiniApps.*

> **✓ Tip**
>
> *When we define the MiniApp we can also specify which MiniApp framework functions we want to use, if any. To display the "Refresh" function in the MiniApp tray, we must set this flag when maintaining the MiniApp. To activate "Personalization," we need to assign an Internet service specifically implemented for personalization. The user can call this service by clicking the "Personalization" button in the MiniApp tray in the Workplace. We do not, however, include this step in our example.*

Until now, we could only execute the service for the MiniApp as a standalone application. In reality, each MiniApp is always part of a role, so we must assign our MiniApp to a suitable role if we want to display it in the Workplace.

To do this, we open "Role maintenance" (transaction PFCG) in the component system and choose a role.

We then choose the "MiniApps" tab and enter the name of the MiniApp definition (Release 4.6D). Next, we choose a priority, which tells the system where in the WorkSpace to display the MiniApp (top, middle, or bottom). To assign the user to the role in the SAP component system, we then choose the "User" tab and enter the user's name.[10]

Figure 13 shows the relevant part of the "Role maintenance" screen (transaction PCFG). The MiniApp definition "CURRENCYCONVERSION" has been added to the role "Employee Self-Service (FI)."

Since the Workplace server and the component system are generally distributed on different SAP systems, we need to import the role from the component system to the Workplace server using an RFC. To do this, we simply choose "Read from other system by RFC" from the initial screen of the "Role maintenance" transaction (PFCG) in the Workplace server. Finally, we assign the user to the role in the Workplace server.

> **✓ Tip**
>
> *After you have defined the MiniApp in "MiniApp maintenance" and assigned it to a role, we advise you to refresh the MiniApp data on the Workplace server. To do this, start the transaction S_WP_CACHE_RELOAD in the Workplace server, choose the destination of the relevant component system, and control the data transfer manually. Otherwise, the system will automatically trigger a data comparison at a specified time.*

Launching the MiniApp in the mySAP Workplace

Now we're ready to launch the MiniApp. We start the Workplace using the correct URL in the browser — for this, we need the name of the Workplace server and the Internet service used to start the Workplace.

[10] In Basis Release 4.6C, MiniApps were added to roles using a URL. For more information on this procedure, see the mySAP Workplace documentation.

Figure 13 "Role Maintenance" in the mySAP Workplace

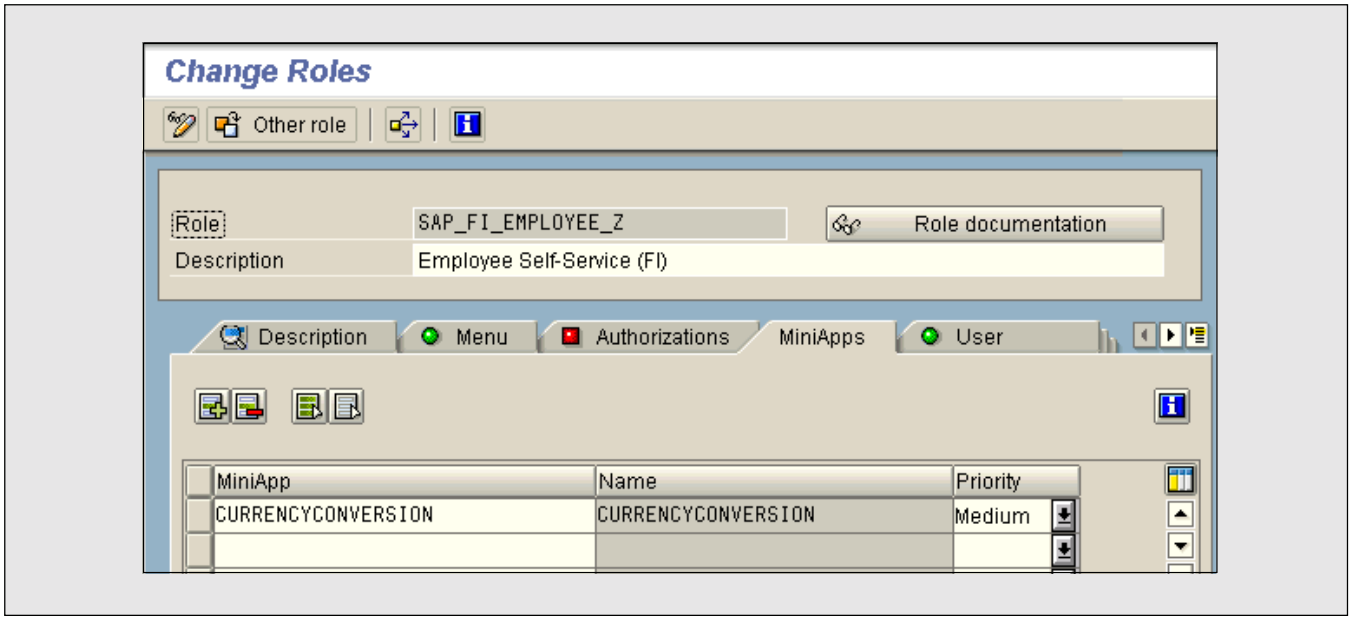
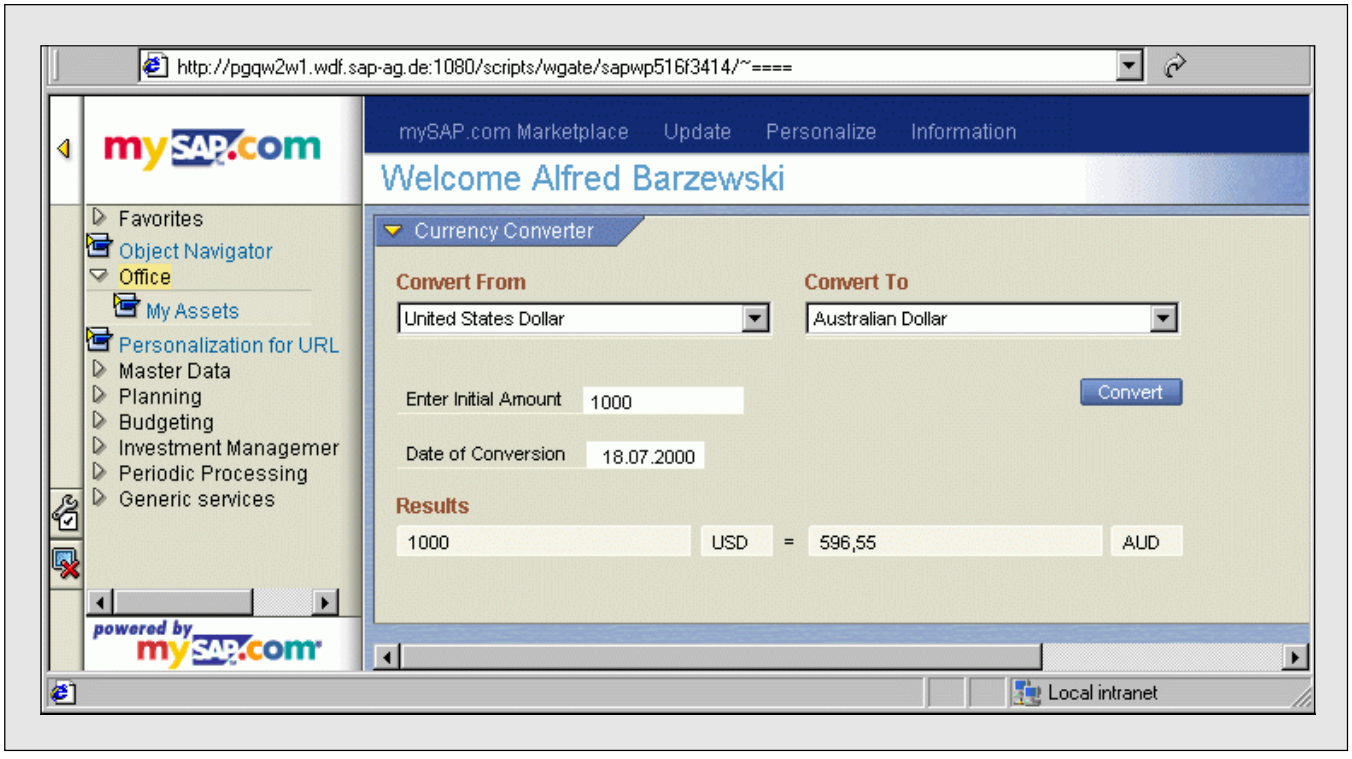


Figure 14 Launching the MiniApp in the mySAP Workplace



We log on in the dialog box that appears. As shown in **Figure 14**, the system then displays the LaunchPad on the left side and our MiniApp in the WorkSpace.

Conclusion

In Release 4.6C, SAP extended the ABAP Workbench development environment to include the Web Application Builder, so that — for the first time ever — developers could use the Workbench to develop ITS-based Web applications, such as MiniApps. This allows developers to take full advantage of the many features of the ABAP Workbench, and to use the Web Application Builder in ways familiar to them. With an improved learning curve, developers can start using the integrated tools and utilities effectively right away, and shift their focus instead to visual and interactive design issues.

Alfred Barzewski received his honours degree in physics in 1991 from the University of Heidelberg, Germany. After working as a college teacher in physics and mathematics, he joined SAP in 1997 as a member of the ABAP Workbench product management group, where he was responsible for online documentation on Remote Communication, non-SAP access to BAPIs, RFC programming in ABAP, and ABAP development tools. He currently works on documentation for SAP's Web Development platforms, focusing on implementation of reusable demo components for ABAP developers. Alfred can be reached at alfred.barzewski@sap.com.

Appendix —
Source Code and Parameters of the Currency Converter MiniApp

This appendix accompanies the article "Ready to Build Your First MiniApp? It's Quick and Easy with the ABAP Workbench!" and contains the presentation source code, the flow logic code, the service parameters, and the theme parameters of the Currency Converter MiniApp described in the article. You can also find this information available for download at **www.SAPpro.com**.

Note that in the following code, the HTML[Business] elements are enclosed in inverted apostrophes (**‘…‘**) and highlighted in bold to distinguish them from standard HTML (normally they are highlighted in blue).

Presentation Source Code

```
‘include(~service="system", ~language="", ~theme="dm",
~name="TemplateLibraryDHTML.html");‘
<html>
  <head>
    ‘SAP_Stylesheet()‘        <!-- inserts standard SAP Stylesheet -->
    ‘SAP_TemplateJavaScript()‘<!-- inserts standard SAP JavaScript functions -->

    <script language="JavaScript">
      function convert()
      {
          document.input_form.elements['~event'].value = 'onConvert';
          document.input_form.submit();
      }

    </script>
  </head>

  <body ‘SAP_TemplateBodyAttributes( )‘ onLoad ="‘SAP_TemplateOnLoadJavaScript()‘">


  <form name="input_form" action="‘wgateURL()‘"  method="post">
  <input type="hidden" name="~event" value="">
```

(continued on next page)

(continued from previous page)

```
<table>
  <colgroup>
      <col width=170>
      <col width=40>
      <col width=10>
      <col width=170>
      <col width=40>
  </colgroup>

  <tr>
      <!--  Label fields for Currency  -->
      <td colspan=3>
      <font face="Arial" size=4 color=#965136> <b>‘#convertfrom‘</b></font>
      </td>
      <td colspan=2>
      <font face="Arial" size=4 color=#965136> <b>‘#convertto‘</b></font>
      </td>
  </tr>
      <td colspan=2>
       <!--   Drop down listbox for "From" currency     -->
      ‘SelectionIndex1 = 1‘
   ‘for ( i = 1 ; i <= CURRENCY_LIST-CURRENCY.dim ; i++ )‘
       ‘if ( LocalCurrencyUnit == CURRENCY_LIST-CURRENCY[i] )‘
            ‘SelectionIndex1 = i‘
            ‘i = CURRENCY_LIST-CURRENCY.dim‘
       ‘end‘
   ‘end‘

      ‘SAP_TemplatePulldownField("FromCurr_ID"
                                 ,name="LocalCurrencyUnit"
                                 ,key="CURRENCY_LIST-CURRENCY"
                                 ,content="CURRENCY_LIST-LONG_TEXT"
                                 ,selIndex=SelectionIndex1
                                 ,width="220")‘
      </td>
      <td></td>
      <td colspan=2 align=right>
       <!-- Drop down list box for "To" currency        -->
   ‘SelectionIndex2 = 1‘
   ‘for ( i = 1 ; i <= CURRENCY_LIST-CURRENCY.dim ; i++ )‘
       ‘if ( ForeignCurrencyUnit == CURRENCY_LIST-CURRENCY[i] )‘
            ‘SelectionIndex2 = i‘
            ‘i = CURRENCY_LIST-CURRENCY.dim‘
       ‘end‘
   ‘end‘
```



```
    ‘SAP_TemplatePulldownField("ToCurr_ID"
                                ,name="ForeignCurrencyUnit"
                                ,key="CURRENCY_LIST-CURRENCY"
                                ,content="CURRENCY_LIST-LONG_TEXT"
                                ,selIndex=SelectionIndex2
                                ,width="220")‘
    </td>
<tr>
<!-- Separator row -->
    <td  height=10 colspan=5></td>
</tr>
<tr>
    <td colspan=2>
    <!--   Input field for initial amount            -->
    ‘SAP_TemplateEditableField("Amount_ID"
                                ,fieldLabel=#amount
                                ,fieldLabelWidth="90"
                                ,name="LocalCurrencyAmount"
                                ,value=LocalCurrencyAmount
                                ,size=15
                                ,maxlength=30
                                ,align="left")‘
    </td>
    <td colspan=3 align=right>
    <!-- Button to perform calculation               -->
    ‘SAP_TemplateLargeActionButton("CalcButton_ID"
                                ,buttonLabel=#convert
                                ,onclick="convert()")‘
    </td>
</tr>

<tr>
<!-- Output/Inputfield for Date  of conversion       -->
    <td align=left colspan=5>
    <!--  Initialize the Date of conversion          -->
    ‘Date = InitialDate‘

    ‘SAP_TemplateEditableField("CalcDate_ID"
                                ,fieldLabel=#calcdate
                                ,fieldLabelWidth="90"
                                ,name="Date"
                                ,value=Date
                                ,size=10
                                ,maxlength=10
                                ,align="right")‘
```

(continued on next page)

(continued from previous page)

```
        </td>
        </td>
    </tr>

    <tr>
    <!-- Separator row -->
    <td  height=7 colspan=5> </td>
    </tr>
    <tr>
        <td colspan=5>
        <font face="Arial" size=5 color=#965136> <b>‘#results‘</b></font>
        </td>
    </tr>
    <tr>
    <!--   output fields for amount                  -->
        <td>
        ‘SAP_TemplateNonEditableField("FromAmount_ID"
                                    ,value=LocalCurrencyAmount
                                    ,size="170")‘
       </td>
       <td>
       ‘SAP_TemplateNonEditableField("FromCurrUnit_ID"
                                    ,value=LocalCurrencyUnit
                                    ,size="40")‘
        <td align=center>
        ‘SAP_TemplateLabel( "LabelEquals_ID"
                                    ,labelText="=" )‘
        </td>
        <td align=right>
        ‘SAP_TemplateNonEditableField("ToAmount_ID"
                                    ,value=ForeignCurrencyAmount
                                    ,size="170")‘
        </td>
        <td align=right>
        ‘SAP_TemplateNonEditableField("ToCurrUnit_ID"
                                    ,value=ForeignCurrencyUnit
                                    ,size="40")‘
        </td>
    </tr>
  </table>
  </form>
 </body>
</html>
```


Flow Logic Code

```
<flow>

 <state name="Start">
      <module name="Currency.GetList" type="BAPI">

         <persistent name="CURRENCY_LIST-CURRENCY"/>
         <persistent name="CURRENCY_LIST-LONG_TEXT"/>

      </module>
 </state>

 <state name="convert">
      <module name="CURRENCY_CONVERSION_MINIAPP" type="RFC">

           <inputmapping source="LocalCurrencyAmount" target="I_AMOUNT_1"/>
           <inputmapping source="LocalCurrencyUnit" target="I_CURRENCY_1"/>
           <inputmapping source="ForeignCurrencyUnit" target="I_CURRENCY_2"/>
           <inputmapping source="Date” target="I_DATE"/>

           <outputmapping source="E_AMOUNT_2" target="ForeignCurrencyAmount"/>
           <outputmapping source="E_DATE" target="InitialDate"/>

      </module>
 </state>

 <event name="onLoad" next_state="Start"/>
 <event name="onConvert" next_state="convert"/>

</flow>
```

Service Parameters

Parameter name	Value
~INITIALTEMPLATE	INITIAL
~LOGIN	
~PASSWORD	
~WEBTRANSACTIONTYPE	MINIAPP
~XGATEWAY	SAPXGINET

Theme Parameters

Parameter name	Value
AMOUNT	Enter Initial Amount
CALCDATE	Date of Conversion
CONVERT	Convert
CONVERTFROM	Convert From
CONVERTTO	Convert To
RESULTS	Results

