# Deployment Macros

**On this Page:**

All backends have an alias definition in the component diagram. So the backend information is hidden from the activity diagrams and access to backends within activity diagrams can be established by using these aliases. An alias links the physical definition of the backend in the component diagram with the backend adapter in the activity diagram. In the activity diagram, the alias is defined by specifying the tagged value **alias** on an action node, which is stereotyped as backend adapter. For more details, please refer to Aliases.

In cases where you need to reference information, which is located in the component diagram and is needed in activity diagrams, the separation of the logical view and the physical view needs to be softened. Suppose you want to access a Web Server to request a specific Web page. Protocol, host, and method are static information that is defined in the component diagram. The URL, which is composed of protocol, host name, and path may contain an id as a dynamic parameter that is known only at run-time. To build such a service, you would define the static information in the component diagram. This information can be accessed from the activity diagram, and can be concatenated with the run-time value of the id.

In activity diagrams, data that reside in component diagrams can be accessed with deployment macros. The deployment macros are used within action scripts of activity diagrams. Aliases help to specify, which data of which backend is to be retrieved. The tagged value **alias** needs to be defined on the action node that contains the deployment macros. In this case, the action node needs not necessarily to be stereotyped as backend adapter.

Deployment macros are expanded to their actual values during compilation. This means that return values of macros are not objects of a basic type but constants. For instance, concat (getURLFromAlias(), "/index.htm") works, whereas getURLFromAlias().concat("/index.htm") does not.

Deployment macros only retrieve the values specified in the component diagram. **Deployment settings, that have been changed via the Bridge, will not been taken into consideration.**

- getCompositeHost() Macro
- getCompositeCategory() Macro
- getCompositeName() Macro
- getCompositeVersion() Macro
- getEncodingFromAlias() Macro
- getHostFromAlias() Macro
- getMethodFromAlias() Macro
- getPathFromAlias() Macro
- getPortFromAlias() Macro
- getProtocolFromAlias() Macro
- getResourceNameFromAlias() Macro
- getResourceTypeFromAlias() Macro
- getURLFromAlias() Macro
- getUserFromAlias() Macro

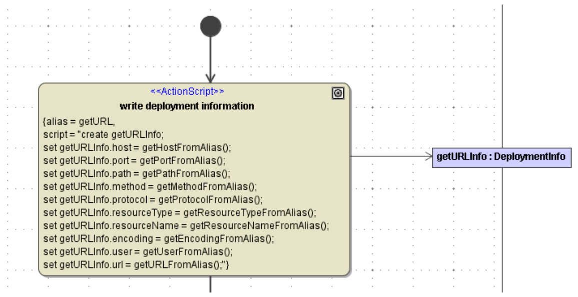
**Example File (Builder project E2E Action Language/Operating):**

<your example path>\E2E Action Language\Operating\uml\deploymentMacros.xml

## UML Example

The following example shows how to get the static information from the deployment diagram, and how to store the retrieved data in an object.

*Figure: Example with Deployment Macros*



```
{alias = getURL,
script = "create getURLInfo;
set getURLInfo.host = getHostFromAlias();
set getURLInfo.port = getPortFromAlias();
set getURLInfo.path = getPathFromAlias();
set getURLInfo.method = getMethodFromAlias();
set getURLInfo.protocol = getProtocolFromAlias();
set getURLInfo.resourceType = getResourceTypeFromAlias();
set getURLInfo.resourceName = getResourceNameFromAlias();
set getURLInfo.encoding = getEncodingFromAlias();
set getURLInfo.user = getUserFromAlias();
set getURLInfo.url = getURLFromAlias();"}
```

In the example above, the alias **getURL** (tagged value of the action **write deployment information**) specifies, from which backend the deployment data will be retrieved (refer to the alias **getURL** that is drawn in the component diagram of the example model **deploymentMacros.xml**). Using the macro **`getPortFromAlias()`** would retrieve the value **80**. Using the macro **`getMethodFromAlias()`** would retrieve the value **get**.

## Deployment Macros in Builder Releases before 6.0

All backends have a physical definition in the deployment and component diagrams. In the deployment diagram, the technical information for each backend like host name, protocol, etc. is defined. This is advantageous, because the information is hidden from the activity diagrams. Access to backends within activity diagrams can be established by using aliases. An alias links the physical definition of the backend in the deployment diagram with the logical definition of the backend adapter in the activity diagram. In the activity diagram, the alias is defined by specifying the tagged value **alias** on an action state, which is stereotyped as backend adapter. The alias is also specified on dependencies in deployment and component diagrams to establish the link to the activity diagram. For more details, please refer to chapter **Aliases**.

In cases where you need to reference information, which is located in the deployment diagram and is needed in activity diagrams, the strict separation of the logical view and the physical view needs to be softened. Suppose you want to access a Web Server to request a specific Web page. Protocol, host, and method are static information that is defined in the deployment diagram. The URL, which is composed of protocol, host name, and path may contain an id as a dynamic parameter that is known only at runtime. To build such a Service, you would define the static information in the deployment diagram. This information can be accessed from the activity diagram, and can be concatenated with the runtime value of the id.

In activity diagrams, data that reside in deployment diagrams can be accessed with deployment macros. The deployment macros are used within action scripts of activity diagrams. Aliases help to specify, which data of which backend has to be retrieved. The tagged value **alias** needs to be defined on the action state that contains the deployment macros. In this case, the action state needs not to be stereotyped as backend adapter necessarily.

Deployment macros are expanded to their actual values during compilation. This means that return values of macros are not objects of a basic type but constants. For instance, concat (getURLFromAlias(), "/index.htm") works, whereas getURLFromAlias().concat("/index.htm") does not.