Tools for Service Refinement
Copyright
Copyright © 2006 Above All Software, Inc. All Rights Reserved.

Restricted Rights Legend
This software and documentation is subject to and made available only pursuant to the terms of the Above All Software Evaluation License Agreement and may be used or copied only in accordance with the terms of that agreement.

Information in this document is subject to change without notice and does not represent a commitment on the part of Above All Software.

Trademarks or Service Marks
Above All Software, Above All Studio, Above All SOA Repository, Above All Knowledge Packs, Above All Multi-channel Deployment are registered trademarks of Above All Software, Inc.

All other trademarks are the property of their respective companies.

Document Code
GST401182007
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before you begin</td>
<td>4</td>
</tr>
<tr>
<td>Scope of this tutorial</td>
<td>5</td>
</tr>
<tr>
<td>Step-by-Step lessons</td>
<td>6</td>
</tr>
<tr>
<td>Lesson 1: Understand service behavior</td>
<td>6</td>
</tr>
<tr>
<td>Lesson 2: Understand service behavior</td>
<td>9</td>
</tr>
<tr>
<td>Lesson 3: Change operation behavior</td>
<td>11</td>
</tr>
<tr>
<td>Lesson 4: Create script operation inputs and outputs</td>
<td>15</td>
</tr>
<tr>
<td>Lesson 5: Create script operation behavior</td>
<td>17</td>
</tr>
<tr>
<td>Lesson 6: Change operation behavior</td>
<td>25</td>
</tr>
<tr>
<td>Lesson 7: Create a script operation</td>
<td>28</td>
</tr>
<tr>
<td>On-your-own Exercises:</td>
<td>32</td>
</tr>
<tr>
<td>Lesson 1: Create a new script operation</td>
<td>32</td>
</tr>
<tr>
<td>Lesson 2: Test the new operation</td>
<td>32</td>
</tr>
<tr>
<td>Lesson 3: Create the composite update operation</td>
<td>33</td>
</tr>
<tr>
<td>Lesson 4: Test the composite update operation</td>
<td>34</td>
</tr>
<tr>
<td>Appendix A</td>
<td>35</td>
</tr>
<tr>
<td>Visual Keys</td>
<td>35</td>
</tr>
<tr>
<td>Toolbar Icon Map</td>
<td>35</td>
</tr>
<tr>
<td>Dictionaries Toolbar Map</td>
<td>35</td>
</tr>
<tr>
<td>Dictionaries Window Component Display Options Map</td>
<td>36</td>
</tr>
<tr>
<td>Complex Object Editor Toolbar Map</td>
<td>37</td>
</tr>
<tr>
<td>Appendix B</td>
<td>38</td>
</tr>
<tr>
<td>Changing Form Options</td>
<td>38</td>
</tr>
<tr>
<td>Appendix C</td>
<td>39</td>
</tr>
<tr>
<td>Above All Studio Script Editor</td>
<td>39</td>
</tr>
<tr>
<td>Script Editor Window</td>
<td>39</td>
</tr>
<tr>
<td>Script Editor Window Toolbar</td>
<td>40</td>
</tr>
<tr>
<td>Above All JavaScript</td>
<td>40</td>
</tr>
<tr>
<td>Appendix D</td>
<td>41</td>
</tr>
<tr>
<td>Adobe Reader Navigation</td>
<td>41</td>
</tr>
</tbody>
</table>
Before you begin

In Getting Started Tutorials 1, 2, and 3 we began the work to create a combined product catalog for parts from our two toy companies – The Perfect Toy and Play Your Game. We have created a complex object type that represents the data we want to manipulate within the product catalog entries. We have created two operations, query and insert, that perform functions on the composite product catalog. In our scenario for the merger of The Perfect Toy and Play Your Game, we chose the database used by The Perfect Toy to carry forward our company product information. The information in this database encompassed all that we need for future products as the businesses merge to form Perfect Play. Getting Started Tutorial 4 assumes that you have used Tutorials 1 through 3 to gain familiarity with the Above All Studio product and its functions and have completed all the exercises in those Tutorials.

Icons are used within this tutorial to highlight key points and assist you in navigating to additional information:

**Visual Key**
These are pointers to graphic descriptions of the screens, icons, and navigation aids in this tutorial.

**Use Adobe Reader Previous View button to return to tutorial**
This will return you back to your place in the tutorial.

**In Depth**
This is a pointer to additional information on the topic.

**Key Information**
This icon marks a text box containing information to assist you in understanding tutorial steps or concepts.

**Decision Point**
This icon indicates a choice of approach or strategy is needed.

**Error Alert**
This icon points out a potential error and provides advice on avoiding it.

**AAS Help: Keywords → Topic**
This icon provides keywords and topics that point to additional information in Above All Studio Help. Use Above All Studio menu bar Help > Search and enter the Keywords provided. Then select Topic from the display list.

This tutorial is provided in Adobe PDF format, intended to be read with the Adobe Reader. Click on the icon for a quick guide to Adobe Reader navigation.
Scope of this tutorial

This tutorial is designed to take you further along the Above All Refine step. Since we don’t live in a perfect world, often we don’t have exactly the services we want to create the composite application we need. This tutorial will demonstrate how we create additional business logic as services in Above All Studio. It will build on the knowledge and expertise we have acquired in the Getting Started Tutorials 1 through 3.

This tutorial will:

- Examine an existing composite service to better understand its behavior
- Create script operations that implement pieces of the business logic that will help us implement the behavior the business needs
- Test the script operations we create to ensure their correct behavior
- Visually refine our composite service using the new script operations to produce the behavior the business needs

This tutorial is designed to take 60 minutes to complete.
Step-by-Step lessons

Lesson 1: Understand service behavior

In the Getting Started Tutorials 2 and 3, we developed the ProductCatalog complex object and composite operations for querying the combined product catalog and creating (inserting) a new entry in the combined product catalog. In this lesson, we are going to look more closely at the exact behavior of the Product Catalog query service. Specifically, we want to determine what the query service returns for a variety of input search values.

We will verify the behavior by experimenting with queries using input values for full and partial ThePerfectToy product numbers and PlayYourGame part numbers. Part and product numbers are formatted in series of codes that organize the data into product types/families and uniquely identify the product and its version. Refer to the AAS Getting Started Tutorial Scenario for in depth information on the PlayYourGame part number format, ThePerfectToy product code format, and the master part number format.

By entering full or partial series of these codes, we can determine whether the operation returns all the information we expect and will need from the query. We will input specific values and examine the results returned to fully characterize the query operation behavior.

1. Click next to the PerfectPlay folder, the ProductCatalog complex object type, and Operations to view the operations for ProductCatalog.

2. Right-click on the QueryProductCatalog operation to open the drop down menu.

3. Select Create New Form to generate a user interface for the QueryProductCatalog operation.

   Depending on the default values in your Above All Studio Tools>Options menu, your new form may have Execute and Cancel buttons at the bottom. All the screen shots in this tutorial standardize on the direct invocation style of form. Click here for information on changing form settings.

4. Maximize the window to optimize the screen real estate.

5. Select Design Mode on the Above All Studio toolbar to enter design mode for the form user interface.

6. Click on the ProductCatalog grid.

7. Click and Drag on the sizing tab at the right of the selected grid to resize the grid so that you can view all of the elements (the last element is Active) at the same time.
8. Select **Run Mode** on the Above All Studio toolbar to change the manipulation mode so that we can test the composite application.
Enter each value in the table below in the appropriate input field of the form, Execute the QueryProductCatalog, and Record the resultant master part numbers found by the query.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
<th>Your Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Part Number</td>
<td>BGO</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROD</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>GGO010</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROD101</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>TGO10001N</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROC20102N</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>TGO01001N</td>
<td></td>
</tr>
<tr>
<td>Product Part Number</td>
<td>GGO0101N</td>
<td>All part numbers</td>
</tr>
</tbody>
</table>

We can conclude the following behaviors from our testing:

- The query will not return matches on partial input values for part numbers from PlayYourGame (BGO, GGO010) or ThePerfectToy (ROD, ROD101)
- The query will not return matches on PlayYourGame full part number input values in master part number format (TGO10001N, TGO01001N)
- The query will return matches on ThePerfectToy full part number input values (ROC20102N)
- The query will return all parts when given no input values

In the next lessons we explore why we got these results and how we can modify this behavior.

9. Click \(\times\) to close the form.

10. Select the No button so that the form is not saved.
Lesson 2: Understand service behavior

In Lesson 1, we noted that we were not getting any values returned when we searched on PlayYourGame parts in master part number format. This behavior occurs because the QueryParts operation for PlayYourGame accepts only full part numbers in the PlayYourGame original format as input. In our merged product catalog in Perfect Play, all references to parts are in master part number format. Instead of using the original PlayYourGame part number format, we need to be able to find PlayYourGame Parts by using master part numbers.

In the Web services that we mined for PlayYourGame, an operation called GetPYGPartNumber was included. In this lesson, we will explore whether that operation may be useful to us in translating a master part number into a PlayYourGame original format part number.

1. Click next to the PlayYourGame folder to view its contents.

2. Double-click on the GetPYGPartNumber operation to open the Operations Editor to view its inputs and outputs.

It appears that this operation accepts a part number in MasterPartNumber format and returns a part number in PlayYourGame PartNumber format. Let's test the operation to confirm its behavior.

3. Click to close the Operations Editor Window.

4. Right-click on the GetPYGPartNumber operation to open up its menu.
5. **Select Create New Form** to generate a user interface for the *GetPYGPartNumber* operation.

![Image of New Form](image)

6. **Enter** each value in the table below in the appropriate input field of the form, **Execute** the *GetPYGPartNumber*, and **Record** the resultant master part numbers found by the query.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Part Number</td>
<td>BGO</td>
<td></td>
</tr>
<tr>
<td>Master Part Number</td>
<td>GGO010</td>
<td></td>
</tr>
<tr>
<td>Master Part Number</td>
<td>TGO10001N</td>
<td></td>
</tr>
<tr>
<td>Master Part Number</td>
<td>TGO01001N</td>
<td></td>
</tr>
<tr>
<td>Master Part Number</td>
<td>TGO01001N</td>
<td></td>
</tr>
</tbody>
</table>

We can conclude the following behaviors from our testing:

- The get operation returns an error message indicating that the entered master part number was not found when no part number match is found (no master part number entered, TGO10001N)
- The get operation returns an error message indicating that the entered master part number was not found when the input values are partial part numbers from *PlayYourGame* (BGO, GGO010)
- The get operation returns matches on *PlayYourGame* input values in full master part number format (TGO01001N)

7. **Click** to close the form.

8. **Select** the **No** button so that the form is not saved.
Lesson 3: Change operation behavior

The business likes the work we have been doing in our integration of Play Your Game and The Perfect Toy product catalogs. However, they need greater flexibility in the Perfect Play product catalog query operation. They need to be able to search on any master part number or partial master part number and get all of the product matches back. As we noted in Lesson 1, the current behavior of the QueryProductCatalog operation will not meet these business requirements.

To meet the business requirements, we will update the QueryProductCatalog operation’s behavior in stages. First, we will update the QueryProductCatalog operation so that we can extend the behavior from full part number queries to partial part number queries for The Perfect Toy products. To do this, we will need to replace the existing QueryProducts operation from ThePerfectToy with a different operation that will perform partial queries.

1. Click next to ThePerfectToy folder, the Products complex object, and Operations to view the available operations for ThePerfectToy Products complex object.

   Refer to AAS Help: Operations ODBC to determine which operations will do a match on partial input strings like ProductCode.
2. **Double-click** on the *QueryProductCatalog* operation in the *ProductCatalog* complex object of *PerfectPlay* to bring up the *Composite Operations Editor*.

3. **Maximize** the window to optimize the screen real estate.

4. **Right-click** on the *Query_Products* operation in the *Composite Operations Editor* to open the drop down menu.

5. **Select Delete** on the drop down menu to remove this operation from the behavior of the composite operation.

   The query behavior for *ThePerfectToy* products has been removed from the operation leaving only the *PlayYourGame* query behavior.

6. **Drag and drop** the *GetVeryFuzzy_Products* operation from the *Products* complex object in *ThePerfectToy* folder onto the *Composite Operations Editor Window*. 
7. **Connect** the following in the *Composite Operations Editor* to create the input and output element correspondences between *QueryProductCatalog* and *GetVeryFuzzy_Products*:

**Inputs:**
- **QueryProductCatalog**
  - ProductName
  - ProductPartNumber

**Outputs:**
- **GetVeryFuzzy_Products**
  - Response (List)
  - ProductCatalog (List)
  - Name
  - ProductCode

**Inputs:**
- **GetVeryFuzzy_Products**
  - Name
  - ProductCode

**Outputs:**
- **QueryProductCatalog**
  - ProductName
  - ProductCode

---

8. **Click** to close the *Composite Operations Editor*.

9. **Select** the **Yes** button to save the changes we just made to the *QueryProductCatalog* operation.
10. **Follow** the steps described in **Lesson 1** to test these updates to the *QueryProductCatalog* operation by entering each value in the table below in the appropriate input field of the form and verifying the results:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Part Number</td>
<td>ROD</td>
<td>6 Robo Dog products returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>MAD</td>
<td>15 Madagascar Pets products returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>MAD101</td>
<td>3 Alex the Lion products returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROC20102N</td>
<td>1 Robo Cat product returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROC10001</td>
<td>3 Robo Cat products returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td></td>
<td>All part numbers returned</td>
</tr>
</tbody>
</table>

11. **Click** ❌ to close the form you created for *QueryProductCatalog*.

12. **Select** the **No** button so that the form is not saved.
Lesson 4: Create script operation inputs and outputs

Now that we have updated the ProductCatalog query to work on partial searches for ThePerfectToy products, we now must turn our attention to fixing the partial searches for PlayYourGame parts. In our Lesson 1 testing, we noted that the QueryParts operation for PlayYourGame only searches on part numbers in PlayYourGame part format. We want to do searches on master part number formats. In our lesson 2 testing, we noted that the GetPYGPartNumber operation only searches for a complete master part number, not a partial one. Though this capability may be useful, it is not what we need. To solve this problem, we are going to have to create custom business logic that will perform the partial master part number search.

There are two ways that we can create the custom business logic we need. External to Above All Studio, we can use our favorite application development tool and language to create a new Web service that performs the custom behavior we need. Alternatively, we can create a script operation in Above All Studio using JavaScript to define the custom behavior we need. The scripting capabilities in Studio are provided for just this type of situation, so we will use them to solve this problem.

We want to create a script operation that takes a partial search string in master part number format and returns an applicable search string in PlayYourGame part number format. If the input string is a full master part number, we want to return the matching PlayYourGame formatted part number. Refer to the AAS Getting Started Tutorial Scenario for additional information or review of the part number formats for PlayYourGame and for the merged PerfectPlay master part number format.

1. Click and Select the New Script Operation option to bring up the New Operations Editor.

2. Click next to Elements for the PerfectPlay ProductCatalog complex object to view the data elements.

3. Drag and drop the MasterPartNumber element on the New Operation1 Inputs in the New Operations Editor to establish a master part number formatted string as input to our script operation.
4. **Select Add Output Element** on the *New Operations Editor* toolbar to create a new data element as output from our script operation.

5. **Type** “SearchString” as the name of the new output from our script operation.

![Diagram of the NewOperation1 editor with Inputs and Outputs sections]

6. **Select Save** on the Above All Studio toolbar to save the new operation definition without closing the editor window.

7. **Select** the *PlayYourGame* folder in the *Save As dialog* as the location for the new operation.

8. **Type** “GetPYGPartNumberForSearch” for the *File name* to name the new script operation.

9. **Select** the *Save* button on the *Save As dialog* to write the definition to the *PlayYourGame* folder.
Lesson 5: Create script operation behavior

Now that we have created the script operation signature (inputs and outputs), we need to create the business logic that defines the behavior of the operation. Our operation will parse the input master part number formatted string and try to match it with known characteristics of the PlayYourGame formatted part number.

1. **Click Open Script Editor** on the Above All Studio toolbar to bring up the *Script Editor Window* where we will create the custom business logic.

[Click here to further explore the Script Editor Window functionality and Above All Studio JavaScript support.]
Note that the Component value is set to our script operation `GetPYGPartNumberForSearch`. You can navigate to the operation's inputs and outputs to set up other related operations using the Scroll Down next to the component name.

The Event value is set to `onInvoke`. `OnInvoke` is one of a set of events that can be used as a trigger for the custom business logic we will be writing.

2. **Select** the Add Action button under the Action List pane to add a new function to the list of actions taken when the `GetPYGPartNumberForSearch` operation is invoked.

For each event associated with the script operation you can orchestrate a series of actions using the Action List. To start, no actions are defined. The Add Action button allows you to create an ordered list of actions you want to be executed for the chosen event. The Add Action dialog allows you to add either a new operation or an existing operation to the action list.

3. **Check** to make sure the Add a new function radio button is selected.
4. **Select** the **OK** button to add an action that calls a new *onInvoke1* function with the input parameter *MasterPartNumber*.

The new function *onInvoke1* is added to the action list and an empty function body for scripting is created in the scripting window pane. You will note that the script editor pulls in the inputs from the signature we created for the *GetPYGPartNumberForSearch* script operation. Our script code for the function will need to return outputs that match the outputs we created in the *GetPYGPartNumberForSearch* operation signature – *SearchString*.

5. **Open** the file *GetPYGPartNumberForSearch.txt* in the *Script* directory where you installed Tutorial 4.

6. **Select** the contents of the *GetPYGPartNumberForSearch.txt* file and **Copy** them to the Clipboard.
7. **Paste** the contents of the clipboard onto line 3 of the Script function onInvoke1 in the Script Editor.

---

String Representation and Manipulation

To use string manipulation operations such as substring, it is necessary to understand how a string is indexed and how the parameters are used.

Characters in a string are indexed from left to right. The index of the first character is 0, the index of the second character is 1, and so forth.

The string property `length` returns the number of characters in the string. Because strings are indexed starting at 0, the character indexed at string length would refer to a character one index beyond the end of the string.

The string method `substring` takes two arguments `start` and `end`. It returns a string formulated from the source string comprised of the characters starting at character position `start` and continuing either to the end of the string (when the `end` argument is not included) or to the character position just prior to `end` of the string (when the `end` argument is included).

**Syntax**

`substring(start[, end])`

Where:
- `start` is the index representing the starting position of the substring to be returned.
- `end` is optional. It is the index of the string just after the last character in the extracted substring.

For more information usage of string methods in Above All JavaScript see [AAS Help: string](#)
and return the corresponding search string in *PlayYourGame* part number format. To do this we make use of string properties and methods supported in *Above All JavaScript*.

1. The property **length** is used to return the length of a string so that we can determine if it is a complete part number or just a partial one. The statement:
   
   ```javascript
   var PartNumLength = MasterPartNumber.length;
   ```
   
   assigns the length of the element *MasterPartNumber* to the variable PartNumLength.

2. We use the **substring** method to examine sections of the string to determine what kind of part number has been input into the function. This method is heavily used to separate the different sections in the *PlayYourGame* and master part numbers.
   
   ```javascript
   var TestPart = MasterPartNumber.substring(0,2);
   ```
   
   This statement captures the first two characters of the product family code that starts off each part number.

3. Finally, we use concatenation to allow us to build strings from other strings.
   
   ```javascript
   MasterPartNumber = MasterPartNumber + "N";
   ```
   
   This statement completes a partial master part number by appending the MasterPartNumber default code for all physical toys, "N", to it.

---

**String Concatenation**

Strings can be combined together through concatenation. Besides using the “+” operator to specify string concatenation (in the example above), there is a **concat** string method that will perform the same function.

The method **concat** concatenates the text of two or more strings and returns a new string.

**Syntax:**

```javascript
concat(string1, string2, . . . stringX)
```

Where:

* string1, string2, . . . stringX are the strings to be concatenated

For more information usage of string methods in *Above All JavaScript* see

*AAS Help: Concat method*
Above All JavaScript extensions give us access to dictionary elements and operations. In order to lookup the PlayYourGame part number equivalent of a master part number, this script must call GetPYGPartNumber, an operation already defined in the dictionary. The access of a dictionary operation is a two step process in JavaScript. First, we set up a pointer to the operation. Then we call the operation providing the argument(s) expected by the operation. Below is an example from our script operation of two statements that perform these steps.

```javascript
var getPYGpart = Dictionary.getOperation2("GetPYGPartNumber");
GetPYGPartNumberResponse = getPYGpart(GetPYGPartNumber);
```

To set up the pointer to a dictionary operation, we use the construct Dictionary.getOperation2. It takes one parameter which is the dictionary pathname to the operation we want to call. You can use either a fully qualified pathname or a relative pathname. Relative pathnames provide more flexibility when copying dictionary contents, so we will use relative pathnames in all of our Dictionary.getOperation2 calls. In this case, we used the pathname "GetPYGPartNumber" because it is a free standing operation located in the same PlayYourGame folder as our GetPYGPartNumberForSearch operation.

For more information on calling an operation already defined in the dictionary see AAS Help: Reference → Referencing Dictionary Objects in a Script.

The GetPYGPartNumber operation takes the structure: GetPYGPartNumber.MasterPartNumber as input and returns GetPYGPartNumberResponse. GetPYGPartNumberResult. The corresponding structures for each of these variables must be set up for the operation to work as expected. The following statements set up the appropriate input parameter structure.

```javascript
var GetPYGPartNumber = new Object();
GetPYGPartNumber.MasterPartNumber = MasterPartNumber;
```

We set up both of the GetPYGPartNumber parameters by using the built-in constructor called Object. The Object constructor allows us to define variables that have structures. These structures are further defined by assignment statements.

For more information on using constructs in Above All JavaScript, see the AAS Help: Constructors → constructor (User-defined) Objects.
From our testing of the `GetPYGPartNumber` operation in [Lesson 2](#), we noted that whenever the operation could not find a match for the input master part number, it would generate an error. We do not want our script operation to be terminated prematurely whenever we make a `GetPYGPartNumber` call. Above All JavaScript provides exception handling capability that will allow us to more gracefully handle errors. In order to maintain control of script behavior when an error is detected, such as no match being found, we have made use of the try/catch construct.

```javascript
try
  {GetPYGPartNumberResponse = getPYGpart(GetPYGPartNumber); }

catch (errmsg)
{
  SearchString = MasterPartNumber;
  return (SearchString);
}
```

In this section of our script, we use any generated error from the `GetPYGPartNumber` operation as an integral part of our operation logic. If an error is generated (this occurs when a match is not found in the Play Your Game data), we know that there is no match and we can return the original search string. Not finding a match is not catastrophic, and we want to continue our script even if a match is not found.

For more information about these exception handling constructs see the [AAS Help: Exception Handling](#).

8. Select ![on the Script Editor Window toolbar to check the script for any syntax errors.](#)

   At the bottom of the Script Editor Window will be displayed the result of the syntax check. Any errors will be displayed in a new pane at the bottom of the Script Editor Window.

9. Click ✗ to close the Script Editor Window.

10. Click ✗ to close the GetPYGPartNumberForSearch Editor.

11. Select the Yes button to save the changes you have made to the operation in the Script Editor.

12. Create a form for the new `GetPYGPartNumberForSearch` operation. Enter each value in the table below in the appropriate input field of the form and verify the results:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Part Number</td>
<td>BGO</td>
<td>BG</td>
</tr>
<tr>
<td>Master Part Number</td>
<td>GGO010</td>
<td>GG010</td>
</tr>
</tbody>
</table>

![Image](#)
<table>
<thead>
<tr>
<th>Master Part Number</th>
<th>TGO01001</th>
<th>TG010100101 (invokes operation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Part Number</td>
<td>GGO02001N</td>
<td>GG020061104</td>
</tr>
<tr>
<td>Master Part Number</td>
<td>BGO0100</td>
<td>BG010</td>
</tr>
<tr>
<td>Master Part Number</td>
<td>ROC201</td>
<td>ROC201</td>
</tr>
<tr>
<td>Master Part Number</td>
<td></td>
<td>the empty string is returned</td>
</tr>
</tbody>
</table>

13. **Click** [X] to close the form.

14. **Select** the *No* button so that the form is not saved.
Lesson 6: Change operation behavior

Now that we have created a script operation that will translate a full or partial master part number into an appropriate part number search string in PlayYourGame format, we can update our ProductCatalog QueryProductCatalog operation to correct our PlayYourGame part search behavior.

1. **Double-click** on the QueryProductCatalog operation in the ProductCatalog complex object of PerfectPlay to open the Composite Operations Editor.

2. **Maximize** the window to optimize the screen real estate.

3. **Drag and drop** the new GetPYGPartNumberForSearch operation from the PlayYourGame folder onto the Composite Operations Editor.
4. **Disconnect** the following in the *Composite Operations Editor*:

   **Inputs:**
   - **QueryProductCatalog**
     - ProductPartNumber
   - **QueryParts**
     - PartNumber

5. **Connect** the following in the *Composite Operations Editor*:

   **Inputs:**
   - **QueryProductCatalog**
     - ProductPartNumber
   - **GetPYGPartNumberForSearch**
     - MasterPartNumber

   **Outputs:**
   - **GetPYGPartNumberForSearch**
     - SearchString
   - **QueryParts**
     - PartNumber

6. **Click** to close the *Composite Operations Editor*. 
7. **Select** the Yes button to save the changes we just made to the *QueryProductCatalog* operation.

8. **Follow** the steps for **Lesson 1** to create a form to test these updates to the *QueryProductCatalog* operation entering each value in the table below in the appropriate input field of the form and verify the results:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Part Number</td>
<td>BGO</td>
<td>10 Board Game parts returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>GGO010</td>
<td>1 Name that Tune part returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>TGO01001</td>
<td>1 Round About part returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>GGO02001N</td>
<td>1 US National Parks Survey part returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>BGO0100</td>
<td>1 Skimania part returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td>ROC201</td>
<td>3 Robo Cat products returned</td>
</tr>
<tr>
<td>Product Part Number</td>
<td></td>
<td>All part numbers returned</td>
</tr>
</tbody>
</table>

9. **Click** [X] to close the form.

10. **Select** the No button so that the form is not saved.
Lesson 7: Create a script operation

Now that our Product Catalog query meets the business’ requirements to be able to search on any full or partial master part number and get all matches from both databases, we need to move on to the next business requirement. The Play Your Game company had a business rule regarding the use of the status flag. Only one version of a part can be available (status/active = True = able to be added to an order) at a time. This business rule was enforced manually by the product manager when he created new product versions. It was not implemented in the InsertPart operation. In ThePerfectToy system, there were no rules regarding use of the active flag for products. In ThePerfectToy multiple versions of products could be available (status/active = True = able to be added to an order) at any one time. The product manager who created the part versions determined whether the status/active was available or not.

The Perfect Play business team noted that a system where multiple versions of products with varying features and prices coexisted would get confusing for the sales personnel as well as for the customer. They decided that relying on the product manager to honor this rule was insufficient. Instead, they want us to implement product catalog management such that only one version of a part/product can be available (status/active = True) at any one time. Furthermore, when a new version of a part/product is created, they want to ensure that only the latest version of a part/product is the available one.

A quick perusal of the products currently available (query Perfect Play’s ProductCatalog for all parts) shows that not all parts/products conform to the new rules. To correct this situation with the current parts/products and to enforce this situation for future products, we would like to implement business logic that reviews the products and deactivates any products that have versions that are earlier than the latest version for the product.

Our business logic will be implemented as a script operation. The operation will take one input value – a master part number. This value can either be an empty string or a full master part number. If the value is an empty string, the operation will search through all products and versions in the product catalog and deactivate any lower numbered part versions that don’t support the business rule. If the input value is a master part number, the operation will search through all versions for the input master part and deactivate any lower numbered part version. In this way, this script operation will provide support for the one active version business rule.

1. **Click New Operation 🌟** on the Above All Studio toolbar and **Select** the **New Script Operation** option to bring up the **New Operations Editor**.

2. **Click** next to **Elements** for PerfectPlay’s ProductCatalog complex object to view the data elements.

3. **Drag and drop** the MasterPartNumber element to the **New Operation Inputs** to establish a master part number formatted string as input to our script operation.

4. **Drag and drop** the MasterPartNumber element to the **New Operation Outputs** to establish a master part number formatted string as output from our script operation.

5. **Right-click** on the **Outputs MasterPartNumber** element for our new script operation to bring up the edit menu.
6. **Select** the *Make element a list* option so that the operation will return a list of *MasterPartNumber* elements.

7. **Select Save** on the Above All Studio toolbar to save the *New Operation* definition.

8. **Select** the *PerfectPlay* folder in the *Save As dialog* as the dictionary destination for the new operation.

9. **Type** “*DeactivateOldProductVersions*” for *File name* as the name of the new script operation.

10. **Select** the *Save* button in the *Save As dialog* to save the new operation in the *PerfectPlay* folder.

11. **Click Open Script Editor** on the Above All Studio toolbar to bring up the *Script Editor Window* where we will create the custom business logic.

   The Component value should be *DeactivateOldProductVersions*.  
   The Event value should be *onInvoke*.

12. **Select** the *Add action* button under the *Action List: pane* to add a new function to the list of actions taken when the *DeactivateOldProductVersions* operation is invoked.

13. **Check** to make sure the *Add a new function* radio button is selected.

14. **Select** the *OK* button to add an action that calls a new *onInvoke1* function with the input parameter *MasterPartNumber*.

   The new function *onInvoke1* is added to the action list and an empty function body for scripting is created in the scripting window pane.  
   You will note that the script editor pulls in the inputs from the signature we created for the *DeactivateOldProductVersions* script operation.  
   Our script code for the function will need to return outputs that match the outputs we created in the *DeactivateOldProductVersions* operation signature – *MasterPartNumber (string List)*.

15. **Open** the file *DeactivateOldProductVersions.txt* in the *Script* directory where you installed Tutorial 4.

16. **Select** the contents of the *DeactivateOldProductVersions.txt* file and **Copy** them to the Clipboard.

17. **Paste** the contents of the clipboard onto line 3 of the Script function *onInvoke1* in the *Script Editor*.  

Sometimes you want to observe the behavior of your script as it executes. The **Key Information** box on this page discusses the use of `Windows.alert` to display messages during script execution.

18. **Select** 📊 on the *Script Editor Window* toolbar to check the script for any syntax errors.

19. **Click** ✗ to close the *Script Editor Window*.

20. **Click** ✗ to close the *DeactivateOldProductVersions Editor Window*.

21. **Select** the **Yes** button to save the changes you have made to the operation in the *Script Editor*.

---

**Getting Status in your Script Operation**

As you write and test your script, you may want to examine values of variables or get simple updates during the script execution to monitor its progress. `Windows.alert` is a means for displaying a message box during your script execution.

`Windows.alert` pops-up a message box containing a message and a confirmation button. The confirmation button must be selected to close the message box and resume operation execution. The alert message box uses "Alert" as its title and displays an exclamation mark as its icon.

**Syntax**

```
Windows.alert(message[, buttonText])
```

Where:

- `message` is a value that is converted to a string and displayed. Complex messages can be created by concatenating strings and other variables together.
- `buttonText` is optional. It is a string containing the text to appear on the button. If `buttonText` is not included, "OK" appears as the text on the button.

For more information on using `Windows.alert` see 📑 **AAS Help: alert method**
22. **Test** the `DeactivateOldProductVersions` operation by creating a form that contains both the `QueryProductCatalog` operation and the `DeactivateOldProductVersions` operation.

- First query the product catalog and note the Active values for the following master part numbers:

<table>
<thead>
<tr>
<th>Master Part Number</th>
<th>Active Flag Setting from Query</th>
<th>Active Flag Setting after Deactivate</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGO07001N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GGO07002N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROC20101N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROC20102N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROC20103N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROD10101N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROD10102N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROD10103N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGO02001N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGO02002N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Next run the `DeactivateOldProductVersions` operation with an empty string as input. This will go through the entire product catalog and reset the active flag for multiple versioned products. Note the Active values for the master parts after the `Deactivate` operation has run.
On-your-own Exercises:
In the On-your-own exercises, you will implement one more script operation and then create a composite operation that will update Perfect Play product catalog entries.

Lesson 1: Create a new script operation
In this lesson you will implement the NextRevMasterPartNumber operation. This operation will be used to create a new master part number with an appropriate version number for an updated product. You will need to complete the following tasks to successfully implement this operation.

- Create a new script operation with the signature:
  - Inputs: MasterPartNumber
  - Outputs: NewRevMasterPartNumber

- Specify the behavior for this operation by adding the onInvoke1 function using the provided NextRevMasterPartNumber.txt script. The business logic for the creation of the new master part number version is provided in the script NextRevMasterPartNumber.txt which you will find in the Script directory where you installed Getting Started: Tutorial 4.

Lesson 2: Test the new operation
Assure that NextRevMasterPartNumber correctly returns a new MasterPartNumber with the appropriate value.

- Create a New Form to test the NextRevMasterPartNumber operation.
- Verify its correct operation using the following inputs:

<table>
<thead>
<tr>
<th>MasterPartNumber Input</th>
<th>NewRevMasterPartNumber Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROD10101N</td>
<td>ROD10104N</td>
</tr>
<tr>
<td>ROC20102N</td>
<td>ROC20104N</td>
</tr>
<tr>
<td>GGO07001N</td>
<td>GGO07003N</td>
</tr>
<tr>
<td>TGO02002N</td>
<td>TGO02003N</td>
</tr>
<tr>
<td>CRP10101N</td>
<td>CRP10102N</td>
</tr>
</tbody>
</table>
Lesson 3: Create the composite update operation

In this lesson you will create a composite operation that will update a product catalog entry in the Perfect Play ProductCatalog. You will need the following information to successfully implement this operation.

- Using the Composite Operations Editor, create an UpdateProduct operation with the following signature:
  - Inputs: an instance of ProductCatalog
  - Outputs: none

- Operationally, this update operation will:
  - Use the NextRevMasterPartNumber operation created in Lesson 1 of these On-your-own Exercises to create a new master part number revision based on the input ProductCatalog.MasterPartNumber.
  - Use the PerfectPlay InsertProduct operation we created in Getting Started Tutorial 2: On-your-own Solutions to insert a new product into the catalog.
    - Make sure that you are feeding it the new master part number version.
    - Decouple the Active data element between the new UpdateProduct inputs and the InsertProducts.ProductCatalog inputs so that updated products will not take the Active element setting from existing products.
    - Make sure that the InsertProducts Active data element is always set to “True” by using the defaultValue property of the Active data element.
    - Make sure that the operation inputs ProductName, MasterPartNumber, ProductFamily, Description, and ListPrice default to required elements by setting their required property in the Data Rules section of the Properties Editor to “True”.
  - Use the DeactivateOldProductVersions operation we created in Lesson 7 to deactivate any earlier product versions based on the NewRevMasterPartNumber.

- Use the Properties Editor by selecting (Collection) next to the operationInvokeOrder property in the General Properties section in the Properties Editor to assure that the execution order of the operations included in the new composite operation is correct. The order must be:
  1. NextRevMasterPartNumber
  2. InsertProduct
  3. DeactivateOldProductVersions
Lesson 4: Test the composite update operation

We can test the composite application for PerfectPlay's ProductCatalog complex object by creating a new form for the complex object. Include the updated QueryProductCatalog and UpdateProduct operations when prompted by the wizard.

- Enter a new price for master part number MAD10201N (Gloria the Hippo).
- Enter a partial master part number to and use QueryProductCatalog to list the related parts.
- Verify that the products in this product family have correctly updated ListPrice, MasterPartNumber, and Active fields.

Because PlayYourGame is a shared Web services resource, it gets reset every night. As a result there may be some inconsistencies noted when a part has been deactivated. ThePerfectToy is a local database and not shared and therefore data from that source will remain as altered by your product updates. Using your update operation, you can correct any inconsistencies you may notice during testing.

For step-by-step instructions for the On-your-own lessons see Tutorial 4 Getting Started Solutions.
Appendix A

Visual Keys

Toolbar Icon Map

Use Adobe Reader Previous View button to return to tutorial

Dictionaries Toolbar Map

Use Adobe Reader Previous View button to return to tutorial
Dictionaries Window Component Display Options Map

Use Adobe Reader Previous View button to return to tutorial
Use Adobe Reader Previous View button to return to tutorial
Appendix B

Changing Form Options

This visual key describes how to change default options for forms generated using Above All Studio. In this example we change from the default form style that uses a Prepare button for each operation and a single Execute button, to a form style using direct invocation by providing a single button for each operation included in the form.

1. **Select** Tools > Options on the Above All Studio menu bar to change the default form settings.

2. **Select** the Authoring tab to choose the new settings.

3. **Click** the Direct Invocation radio button in the Form Style panel to choose a single invoke button for the operation.

4. **Click OK** to close the Options Window.

All forms created subsequent to this change of options will follow the new style default.

*Use Adobe Reader Previous View button to return to tutorial*
Appendix C

Above All Studio Script Editor

Script Editor Window

The Script Editor Window is divided into sections: the Actions navigation panel which displays the selected component, event and actions list and the Script panel in which the scripting is done. The Navigation panel provides for the selection of Action Components and Events and allows us to Add, Edit or Remove actions from the list. The Script Panel is the pane in which JavaScript is entered for the function.

For more information see AAS Help: The Script Editor

Use Adobe Reader Previous View button to return to tutorial
Script Editor Window Toolbar

The Script Editor Window toolbar provides icons which allow you to customize the appearance of the window and assist in the creation of the script. Window appearance icons allow you to modify panel positioning and show or hide the script and output panels. Scripting icons provide for adding and removing comments, indenting and unindenting code, adding and removing bookmarks, as well as syntax checking.

![Script Editor Window Toolbar Image]

For more information on Above All Java Script see AAS Help Contents: Reference

Above All JavaScript

Above All JavaScript is compatible with ECMAScript except where it has been necessary to add, change, or delete features to better support the web services and XML. Since ECMA script was designed to have limited functionality, additional types are needed to properly support web services. Above All JavaScript has added data types to support the XML types.

For more information on these differences see AAS Help: JavaScript vs ECMAScript

Use Adobe Reader Previous View button to return to tutorial
Appendix D

Adobe Reader Navigation

There are several ways to navigate this document using Adobe Acrobat Reader. The left side of the window contains the navigation pane which displays tabs to provide navigation methods. We have created bookmarks which can be used as a table of contents allowing the user to navigate directly to a section of interest.

The buttons at the bottom of the document pane allow navigation through the Tutorial as does the vertical scroll bar at the right.

This document also contains links within the text and graphics that will take you to locations within the document, to other documents on the web, or to help topics for additional information. As you mouse over areas of the document with links, the cursor shape will change to a pointer to indicate a link is defined at that place in the document.

Use Adobe Reader Previous View button to return to tutorial