



---

## **UML Essentials Localization – Double Byte String Support**

Version 1.2  
November 7, 2002

---

### *PathMATE Technical Notes*

Pathfinder Solutions LLC  
33 Commercial Drive, Suite 2  
Foxboro, MA 02035 USA  
[www.PathfinderMDA.com](http://www.PathfinderMDA.com)  
888-662-7284

# Table Of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Goals.....</b>	<b>1</b>
<b>3. Strategy .....</b>	<b>1</b>
Analysis Models.....	1
<i>Analysis Elements and Fields .....</i>	<i>1</i>
<i>Modeling Conventions .....</i>	<i>1</i>
<i>Analysis Examples.....</i>	<i>2</i>
UML Essentials Features.....	5
<i>Editor Integration.....</i>	<i>5</i>
<i>Springboard Extensions .....</i>	<i>5</i>
<i>Report Template Extensions .....</i>	<i>5</i>
<i>Code Template Extensions.....</i>	<i>5</i>
<b>Appendix A – Analysis Element String and StringList Fields .....</b>	<b>6</b>

## 1. Introduction

This Technical Note describes the approach to be applied to support localization requiring double-byte string support. It covers the impacts on all user-visible aspects of UML Essentials, and how these capabilities are to be applied by the user.

## 2. Goals

Currently UML Essentials integration and tooling has no specific support for double-byte strings, forcing modelers to use a roman alphabet. The purpose of the localization capabilities is to allow modelers to express themselves in local language and gain the full benefit of abstraction. This will include Japanese language support. An additional goal is to constrain the bulk of the impact of the localization changes within Springboard and other "upstream" elements, and reduce impact on any report or code generation templates. If changes to existing report and code templates can be held to a minimum and made in a locale-independent manner, then the same UML Foundation products can be used for all customers regardless of locale, increasing product quality and reducing feature release latency.

## 3. Strategy

### Analysis Models

#### *Analysis Elements and Fields*

Each model artifact is called an analysis element. Examples of analysis elements include domain, class, attribute, state, action, and PAL statement. Analysis elements have fields that convey information about the analysis element. Examples of analysis element fields include domain.name, class.description, and attribute.dataType. A complete list of all analysis elements and fields supported in UML Essentials is in the Springboard User's Guide "Appendix A: Analysis Element Field Reference", starting on page 23 of that document.

The primary strategy has two elements. The first is to simply allow multi-byte values for all analysis element String fields. The second is to allow the use of local language in the specification of "naming fields" along with the concurrent use of a ImplementationName property used to capture a double-byte identifier to be used in code generation for implementation language identifiers.

Please refer to Appendix A of this document for a table of analysis element fields of String type that will support double-byte values.

#### *Modeling Conventions*

The modeler will use local language for analysis element names and descriptions, providing this information in the standard UML editor fields. In addition, the UML Essentials ImplementationName coloring property may be used to specify an English name for any analysis element that has a name field. This ImplementationName may be specified using fields in Rose, or an external property.txt file may be used to specify these using this standard code generation property mechanism.

While MBSE conventions call for a prefix to be specified for the Domain and the Class, their names are still used to construct some implementation-level identifiers, so an ImplementationName is required for any domain or class with a local (double byte) name. All analysis elements listed in the table in Appendix A require an ImplementationName when a local (double byte) name is used, except for those analysis elements with no name: Action, BinaryRel (name is derived), Bridge, NameValuePair, and SubSuperRel.

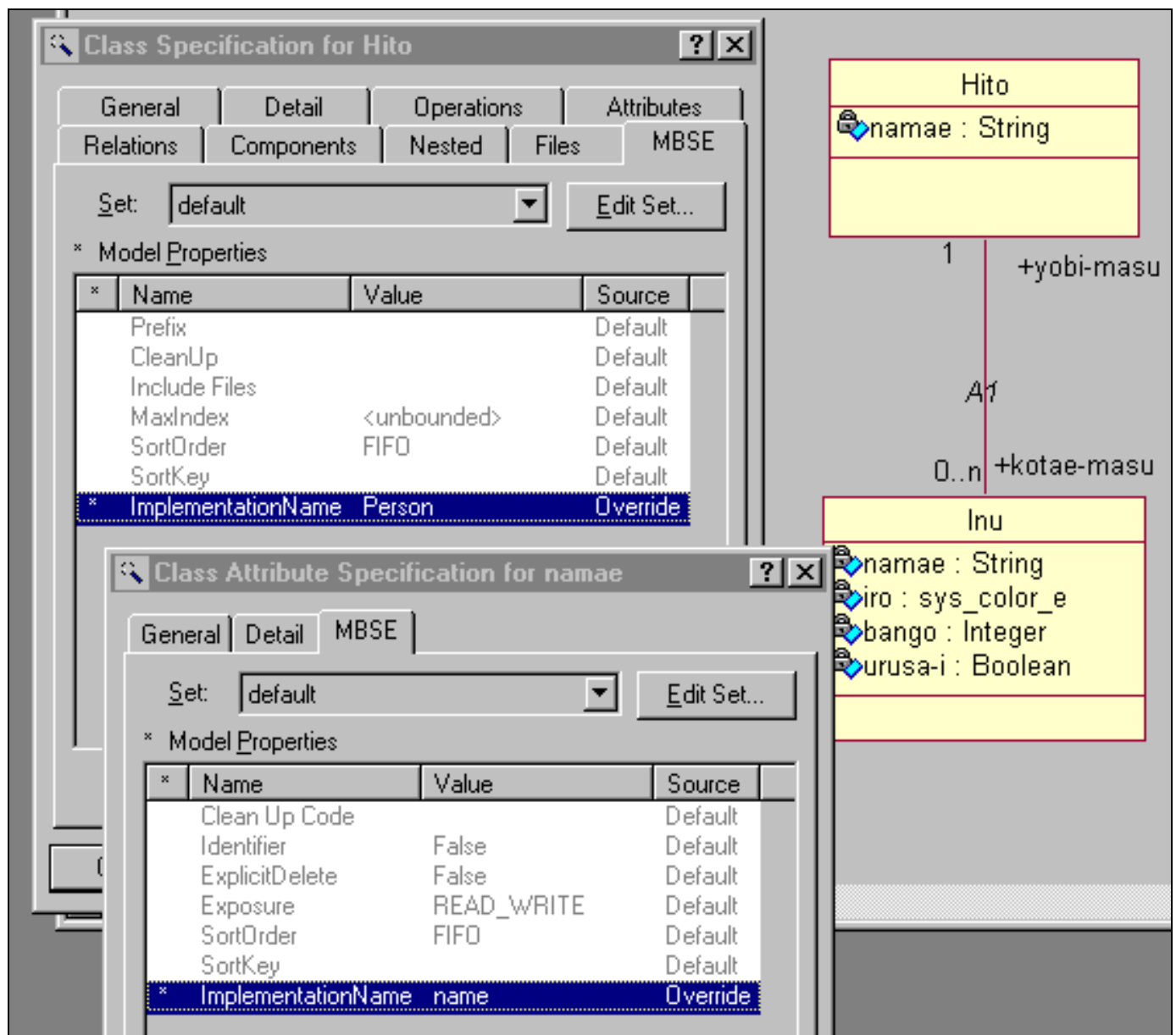
The new system-level property "AutomaticImplementationName" (default FALSE) can be set to TRUE to indicate that an ImplementationName should be automatically constructed by springboard for every analysis element.

### ***Analysis Examples***

Please note that the examples below use roman characters for local language names instead of the appropriate local double-byte characters to be actually used.

#### **Class Diagram**

In the following class diagram example, the class name, attribute names, and association role phrases are all expressed using double-byte local language, and therefore all require ImplementationName values:



The above example only shows two property dialogs, but all of the analysis elements must have ImplementationName specified:

<b>Analysis element</b>	<b>local name</b>	<b>ImplementationName</b>
class	Hito	Person
attribute	namae	name
class	Inu	Dog
attribute	namae	name
attribute	iro	color
attribute	bango	number

attribute	urusa-i	noisy
participant (role phrase)	yobi-masu	calls
participant (role phrase)	kotae-masu	answers
<i>(Apologies for any defective Japanese syntax.)</i>		

Assuming the context of a Pets domain with the domain prefix PETS, the above analysis would generate code fragments looking like:

```
struct PETS_Person
// from class Hito
{
    SW_String name; // from namae
    struct SW_BaseList across_A1_to_answers;
    . . .
```

```
struct PETS_Dog
// from class Inu
{
    SW_String name; // from namae
    sys_color_e color; // from iro
    int number; // from bango
    bool_t noisy; // from urusa-i
    PETS_Person_handle_t across_A1_to_calls;
    . . .
```

### Action Language

The analyst developing PAL actions must be careful to properly mix local language analysis element identifiers and comments with PAL keywords and other fixed syntactic elements in single-byte strings only. Action local variables can be named in local language using the standard PAL statement coloring syntax.

#### **PAL local variable declaration example:**

```
Integer inu_bango; {ImplementationName = "dog_number"}
```

In the above example this PAL local variable is called "inu\_bango" in PAL, and "dog\_number" in the corresponding generated implementation code.

#### **Generated C local variable declaration example:**

```
int dog_number; /* inu_bango */
```

In places where the local variable is used, the local name can be used. At translation time these references will be automatically converted using the `ImplementationName` specified in the local variable declaration.

#### **PAL local variable reference example:**

```
inu_bango = inu_bango + 1;
```

#### **Generated C local variable reference example:**

```
dog_number = dog_number + 1;
```

Comments in PAL actions can be double-byte, and will be carried through to generated target documents.

## **UML Essentials Features**

### ***Editor Integration***

Each of the UML Essentials editor integration support utilities will be extended to use double-byte strings to access and manipulate analysis element String fields. The names of the PAL files will use the local language names of the analysis elements it comes from. The editor extract utility and its accompanying XMI file emitter will also be extended to support double byte local language in naming and description fields as well as all coloring properties, either pre-defined or user-defined. Please note that many of these coloring fields are used to capture information that must be double-byte-only, such as domain and class prefix, relationship sort-order, etc.

### ***Springboard Extensions***

Springboard will be extended to support double byte local language in all String fields including all coloring properties. The Pathfinder Action Language parser in Springboard will be extended to support double-byte input.

For each analysis element with a name field, a companion langId field is already defined to be a copy of the name field cleaned up for use as an implementation language identifier. Springboard will be extended to override the langId value with the `ImplementationName` if it is specified. When the system-level property "AutomaticImplementationName" is found to be TRUE, Springboard will automatically construct a single-byte langId value for every analysis element that does not have one specified via `ImplementationName`.

Please note the Springboard template notation will not be extended to support the specification of double-byte string literals. This limitation may affect direct comparisons between String fields and template literals.

### ***Report Template Extensions***

The report template will be extended to display langId as well as the analysis element name. If a langID distinct from the element name is provided (in the case `ImplementationName` is used) this will be included in the report.

### ***Code Template Extensions***

The code generation templates for UML Foundation for C++, C, and Java will be changed to use the langId field instead of the name field when constructing identifiers for any analysis element. The local analysis element name will be displayed as a comment supplementing the description comments.

## Appendix A – Analysis Element String and StringList Fields

The table below lists all the String-based fields which will support double-byte values. A complete list of all analysis elements and fields supported in UML Essentials is in the Springboard User's Guide "Appendix A: Analysis Element Field Reference", starting on page 23 of that document.

<b><u>Analysis Element</u></b>	<b><u>field</u></b>
Action	actionText
Attribute	description name
BaseType	name
BinaryRel	description name
Bridge	description
DataType	name
Domain	description im name supportDiags
DomainService	description name
Event	description name
MethodInvocation	name
NameValuePair	paramName
Object	description name std
ObjectService	description name
Parameter	description name
Participant	name
Relationship	description
Service	description name
ServiceHandle	name
State	actionSummary description name
SubSuperRel	description
System	description domainChart



	name
	supportDiags
UserDefinedType	name
UserEnumerate	name
UserNonEnumerate	name
VariableDefinition	name