





Business Tasks – Tutorial and Best Practices

VoiceObjects 10.0

To ensure that you are using the documentation that corresponds to the VoiceObjects software you are licensed to use, compare this version number with the software version shown in the Help menu of the VoiceObjects software you are using.

Copyright

Copyright © 2001-2011 Voxeo Germany GmbH and its licensors. All rights reserved.

Published in Germany – Legal information 2011

Information in this document is subject to change without notice and does not represent a commitment on the part of Voxeo or any of its affiliated companies (collectively “Voxeo”). The software described in this document is furnished solely under a license agreement. The software may be used or copied only in accordance with the terms of the license agreement. You shall not reverse engineer the software or sub-license, distribute or make the software available to third parties by other means except as specifically allowed in the license agreement or by mandatory law. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Voxeo.

Protected by German, European and US patents. Further patents pending.

Companies, names, and dates used in examples herein are fictitious unless otherwise noted. If such names affect copyrights or trademarks or others, please notify Voxeo by e-mail at support@voxeo.com.

Trademarks

VoiceObjects is a registered trademark. Any other trademarks, trade names or service marks mentioned in this document belong to their respective owners.

The material presented herein is based upon information that we consider reliable, but we do not represent that it is error-free and complete. Voxeo is not making any representation or granting any warranty with respect to such material, and the distribution of such material shall not subject Voxeo to any liability.

Third-Party Products

If Licensed Products are distributed together with third-party software or if this is contained in the Licensed Products, which are subject to additional license provisions, Licensee undertakes to observe the license provisions of the third party.

Explicit Copyright Notice

The VoiceObjects software includes software developed by the Apache Software Foundation (www.apache.org). Copyright © 1999-2011 – The Apache Software Foundation. All rights reserved.

Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both.

Specific versions of the VoiceObjects software contain copyright material authorized by the Eclipse Foundation (www.eclipse.org), their contributors and others. All rights reserved.

Specific versions of the VoiceObjects software work with Microsoft Excel or make use of copyright material from Microsoft Corporation (www.microsoft.com). All rights to such copyright material rest with Microsoft. Microsoft and Excel are registered trademarks of Microsoft Corporation.

The VoiceObjects software is bundled with the software *NuGram IDE Basic Edition* developed by Nu Echo Inc. (www.nuecho.com). All title and copyrights in and to the software *NuGram IDE Basic Edition*, the accompanying printed materials, and any copies of the software are owned by Nu Echo Inc. and its suppliers.

Document Number: E-041-20110404-VO10



Table of Contents

INTRODUCTION	4
1 MOTIVATION FOR THE BUSINESS TASK OBJECT	5
2 BUSINESS TASKS IN A NUTSHELL	6
Defining a Business Task	6
Manually Starting and Stopping Business Tasks	7
Business Task Completion Report	8
3 DESIGNING BUSINESS TASKS	9
Questions to Be Asked	9
Business Task Design - Best Practices	9
4 IMPLEMENTING BUSINESS TASKS	10
Completion Status: Complete	10
Completion Status: Incomplete.....	10
Status: Incomplete - Session termination.....	10
Status: Incomplete - Recognition failure.....	11
Status: Incomplete - Back-end error.....	12
Status: Incomplete - Technical error	12
Status: Incomplete - Caller abort.....	13
Status: Incomplete - Business logic	14
Status: Incomplete - Task restart.....	14
Business Task Status - Summary	15
5 CONTROLLING BUSINESS TASKS: A HANDS-ON EXAMPLE	16
First Option	16
Second Option	17
Third (and recommended) Option	18
The Business Task Session Analysis Report	19
6 CUSTOM LOGGING WITH BUSINESS TASK DATA	20
Using Business Task Data.....	20
7 WRAPPING UP	23
Stacking Tasks	23
Reporting on stacked tasks	24
Finish task options.....	24
Stacking tasks – best practices	24
Business Task Expressions.....	25
Input State-Level Reports with Business Tasks	25
8 FURTHER INFORMATION	26



Introduction

The *Business Tasks – Best Practices* document explains how to design, implement and control business tasks to easily measure attempt rates and completion rates of tasks from a business – or customer – perspective.

- **Chapter 1 – *Motivation for the Business Task Object*** describes why the Business Task object originally was introduced into the VoiceObjects development environment.
- **Chapter 2 – *Business Tasks in a Nutshell*** shortly describes how to define a Business Task object in VoiceObjects and how to start and stop business tasks through Expression objects.
- **Chapter 3 – *Designing Business Tasks*** describes some best practices around business task design.
- **Chapter 4 – *Implementing Business Tasks*** discusses the possible results of a business task execution and how to handle them in a call flow.
- **Chapter 5 – *Controlling Business Tasks: A Hands-On Example*** shows three different ways of designing a specific business task.
- **Chapter 6 – *Custom Logging with Business Tasks Data*** describes how to make use of business task data.
- **Chapter 7 – *Wrapping-Up*** discusses some additional business task-related features.
- **Chapter 8 – *Further Information*** provides additional sources for further information on business tasks in VoiceObjects.



1 Motivation for the Business Task Object

The **Business Task** object was introduced to the object library of VoiceObjects to meet a very common requirement: Having an easy tool to measure attempt rates and completion rates of tasks from a business – or customer – perspective. A “Task” can for example represent an identification and validation process, a self-service activity such as a money transfer or topping up a prepaid account, or getting a certain piece of information in the IVR (or on any other self-service channel such as mobile web or texting).

We had observed that IVR developers routinely added “log points” to their IVR call flow in order to measure what parts of their application were visited by callers and to create statistics on task completion rates. Doing so, however, was a tedious and error-prone process, and it didn’t get any easier to get it right when the application requirements kept changing. Another “traditional” way of measuring task-related metrics was to get the recognition and hang-up statistics for each individual input state, and try to derive the required metrics from there. Not easy to do, either. So, how nice would it be to get these metrics – the most important ones from a business perspective – more or less out-of-the-box?

The way VoiceObjects handles **Business Tasks** makes it easier than ever to instrument your call flow with minimal effort, by indicating start and success vs. failure criteria for business tasks. VoiceObjects Server will automatically populate the Infostore database with business task related metrics, and you can use standard reports in VoiceObjects Analyzer to analyze the performance of your business task implementations and to create the exact reports that your business stakeholders are asking for.

For details on the configuration of the Business Task object refer to *Business Task* in the *Object Reference*.

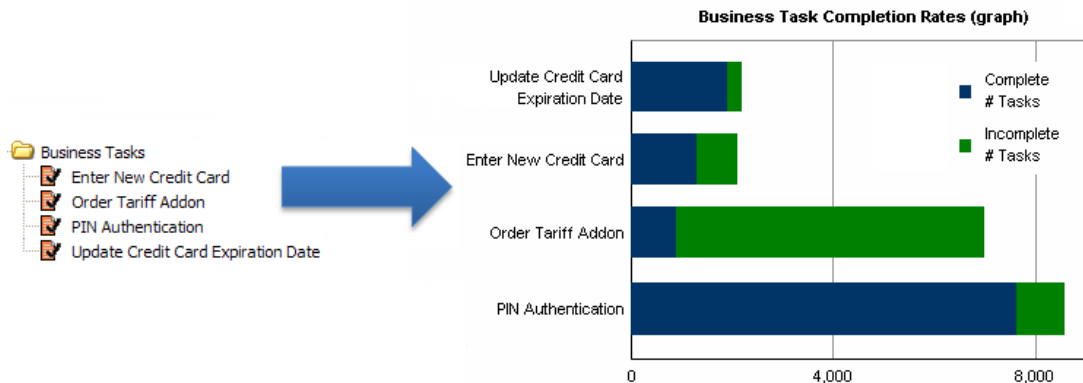


2 Business Tasks in a Nutshell

Defining a Business Task

As we have seen in the previous chapter, a business task is represented by a Business Task object in VoiceObjects.

When creating a Business Task object in order to meet a certain reporting requirement, the most important decision you need to take is how to call it. This is because later, when looking at the VoiceObjects Analyzer reports, the metrics associated with your Business Task object will be identified by that object's name. This is illustrated by the two images below taken from the *Prime Telecom* sample application. The first one shows the Business Task objects defined in the call flow listed in the **Object Browser** in Desktop for Eclipse, the second one shows the *Business Task Completion Rates* standard report in VoiceObjects Analyzer.



As you can see, for each of the Business Task objects defined in the call flow, you get a bar in the chart indicating how many caller attempts to perform a certain task were successful (*Complete*) and how many failed (*Incomplete*). This is the most important – albeit not the only – metric you will get by defining business tasks in the call flow.

Apart from the name, you should define a proper task **Type** and a **Reference ID**. Note that manually assigning the **Reference ID** has the advantage that it will later be easier to use the TASKREFID() function through the Expression object – this function returns the reference ID of the task that is currently active.

Assigning the proper task type is a matter of courtesy to the business analysts who will look at the reports later – it doesn't have any functional impact other than the ability to group task-related results by type.

These are the task types you can select from:



Task Type	Description
Authentication	Any kind of identification, authentication, or verification task, such as entering an account number and PIN at the beginning of a call.
Information	Any kind of information that is presented to the caller, such as playing back the account balance in a phone banking application.
Transaction	Any kind of transaction task, such as booking a flight, transferring money, ordering a product, changing credit card details, etc.
Routing	Any kind of routing task, such as finding out about the reason for calling and transferring the caller to the proper agent group.
Notification	Any kind of external notification sent to the caller, such as email, SMS, MMS, etc.
Other	A task that does not fit in any of the other task types.

Manually Starting and Stopping Business Tasks

Now, let us see how to start and stop a business task in a call flow. Let us have a look at the *Update Credit Card Expiration Date* module from the *Prime Telecom* sample application. Note that there is a START Expression object right at the start of that module's sequence. The STOP function is called only under certain conditions:

- The caller successfully entered an expiration date;
- (s)he confirmed the new expiration date in a subsequent input state; and
- the new date was successfully written to the database.

To start and stop business tasks in a call flow, the *STARTTASK* and *FINISHTASK* functions in the Expression object are used. Below see an example of a *FINISHTASK* function; the first argument is the Business Task object, and the second argument is the status to be assigned on finishing the task:



Definition (...)

Name:

Declaration (...)

Select the desired function and assign a valid object or a constant for each required argument.

Function: ▼

FINISHTASK (task, status, clearNavigationHistory) -
Finishes the Business Task that is specified in the first argument with the status specified in the second argument. This can be one of backEndError, businessLogic, callerAbort, recognitionFailure, technicalError, or complete. Optionally, the first argument can be "all" to finish all tasks

Argument [1]:

Argument [2]:

Stopping a task with the *FINISHTASK* expression is called “manual” termination of the task. We’ll talk about “automatic” methods of task termination in a minute, in the context of task failures induced by reasons such as failure of back-end access, failure of speech recognition, or the caller hanging up in the middle of the conversation.

Business Task Completion Report

VoiceObjects Analyzer provides different task-related reports. The most important one is arguably the *Business Task Completion* report. We already saw the chart in the image above; let’s now have a look at the associated table (or grid):

Task	Task Type	Complete	Incomplete			Total
		Complete	Caller abort	Recognition failure	Session termination	Total
		%	%	%	%	%
Update Credit Card Expiration Date	Transaction	87.2%	1.7%	4.4%	6.7%	100.0%
Enter New Credit Card	Transaction	61.9%	6.2%	15.6%	16.3%	100.0%
Order Tariff Addon	Transaction	12.6%	82.2%	3.5%	1.6%	100.0%
PIN Authentication	Authentication	88.9%	1.4%	3.3%	6.4%	100.0%

For each task, you get the relative numbers of *Complete* vs. *Incomplete* task execution attempts. For the *Update Credit Card Expiration Date* task, for example, you see that 87.2% of all caller attempts to update their credit card expiration date were eventually successful. Note that this image shows a simplified version of the report. The full standard report shows both relative and absolute counts.

The *Incomplete* attempts are further broken down into different failure categories (*status*), such as *Caller Abort*, *Session Termination*, *Recognition Failure* etc. It is very important to get a solid understanding of these different failure categories, so we’ll review each status in some detail further below.



3 Designing Business Tasks

When designing business tasks, you need to understand exactly what reports and metrics business stakeholders, management, and/or customer care want to see.

Questions to Be Asked

The most important **questions** to ask those business stakeholders are:

- What are the tasks that a caller can or needs to accomplish in the self-service application, from the caller's perspective, end-to-end? Such as "login and authenticate", "get product pricing info", "money transfer", etc.
- For each task, when exactly in the call flow do we know that the caller has started with this task?
- For each task, when exactly in the call flow do we know that it can be counted as "successful"? This may include, for example, successfully providing input in several input states, a successful back-end transaction, an explicit confirmation by the caller, and maybe also requires the caller to listen through some full information prompt – there are many details to consider.
Bottom Line: important to get a solid understanding of a "successful" task execution. Also, be aware that sometimes there are multiple ways to achieve the same thing, such as different ways of caller authentication, different ways of topping up an account. Hence, a single task may have multiple entry and exit points in the call flow.
- Are there any task related data that need to be reported on? For example, in the context of a task "money transfer" you would want to get statistics on the currency amount transferred; or in the context of a task "agent transfer", you would want to get a breakdown by agent skill groups.

Business Task Design - Best Practices

These are some important **best practices** around business task design:

- A business task should represent a complete task from the caller perspective. So, when an input state is followed by a confirmation prompt, that confirmation prompt needs to be in the scope of that task. In the case of an information enquiry, the task should only be finished after the caller listened through the information prompt. In the case of a routing task, the business task scope should include both the menu navigation and the (successful) call transfer. And so on.
- If you find that a business task only involves a single input state, question is whether you really need to define a Business Task object in the first place. For each input state, VoiceObjects automatically gathers information such as successful recognition rates, NoMatch/NoInput rates, hangup rates etc anyway, so you might risk creating extra log data without any additional benefit.
Bottom Line: Before designing business tasks, it's advisable to have a good understanding of the input-state level reporting that VoiceObjects Analyzer provides out-of-the-box.
- While it is technically possible to "stack" tasks – i.e., have more than one task active at the same time, so as to have a "super-task" and several sub-tasks – it is not advisable. Try to keep business task definitions clean and simple. One task at a time. This will make life easier both for developers and for the business analysts who read the reports.



4 Implementing Business Tasks

By now, you know how to define a Business Task object, and how to manually start and stop it using functions in Expression objects. Let us now discuss all the possible results of a task execution – whether it was successful, or whether it failed for one reason or another – and how to handle them in the call flow.

The discussion will be structured along the possible task termination status values described below.

Completion Status: Complete

When a task has been successfully completed, the status will be *Complete*. It needs to be manually terminated in the call flow, using the expression function `FINISHTASK(task, "complete")`.

It matters where exactly you place that expression into your sequences. You could put it before or after a final confirmation prompt is being played; before or after a notification SMS message is sent to the user; before or after some required post-processing is done. These implementation decisions depend on business decisions (coming back to the question of what exact requirements a “successful” task execution has to meet) and will have direct influence on task completion statistics.

Completion Status: Incomplete

We will now discuss the different possible status values of a failed (*Incomplete*) business task execution. If a business task execution is *Incomplete*, it will have one of the following status values:

Task Status	Task was finished...
Backend error	...because of an error in a back-end system.
Business logic	...due to requirements or specifications of the business logic.
Caller abort	...due to some caller action (such as “zero out” to an agent).
Recognition failure	...due to speech recognition problems.
Session termination	...because the session was terminated while the task was still open.
Task restart	...because a task was started while a task of the same type was still active.
Technical error	... due to a technical error.

Status: Incomplete - Session termination

When the VoiceObjects session terminates and there are still tasks that are active, these will be finished and automatically marked as *Session termination*. This is the typical scenario when the caller simply hangs up in the middle of the call, before the task was fully completed.



So, when reading task completion reports, *Session termination* usually really means “Caller hangup”.

Note that a large rate of *Session termination* tasks *may* also indicate that you haven’t properly captured all possible successful task terminations. Missing to call the FINISHTASK(task, “complete”) function in your call flow results in the task remaining active, being automatically terminated when the session terminates, and hence being flagged as *Session termination*. The same can happen if you get uncaught events that end the call.

Finally, if you define your own disconnect handling by implementing an event handler for the events *Disconnect*, *Disconnect – Hangup*, or *Disconnect – Transfer*, you want to select *All* in the **Finish Tasks** field of the event handler, as displayed in the following screenshot:

Event	
Label:	<input type="text"/>
Layer:	<input type="text"/>
Channel:	Voice <input type="button" value="v"/>
Occurrence:	If >= 1 <input type="button" value="v"/>
Event type:	Disconnect <input type="button" value="v"/>
Object:	<input type="button" value="↑"/> After Disconnect Handling
Continuation:	Proceed <input type="button" value="v"/>
Finish Tasks:	All <input type="button" value="v"/>

This is our first example of “**automatic**” **task termination** that you need to take care of.

What this configuration does is this: When a *Disconnect* event is being triggered and the according event handler is activated, VoiceObjects Server will check whether there are any open tasks; those will now be automatically terminated and marked as *Session termination*.

Note that a *Disconnect* event handler is typically defined only once, in the main Module object of your application, from where it will be automatically inherited by all dialog components. Hence, there should be only a single place in your application where you need to take care of this setting.

Status: Incomplete - Recognition failure

A common IVR design requirement defines that a caller should be transferred to a live agent after a certain number of *NoMatch* or *NoInput* events. A very common definition is that this happens after 3 failed input attempts in any given input state.

In VoiceObjects call flows, this behavior is usually implemented as an *ASR – Invalid Answer Limit* event handler that is defined, only once, in the main Module object and hence inherited by all dialog components. When defined as in the following figure, this event will be triggered after a sequence of 3 *NoMatch* or *NoInput* events in an input state:



Event	
ASR - Invalid Answer Limit (3)	
Label:	<input type="text"/>
Layer:	<input type="text"/>
Channel:	Voice <input type="button" value="v"/>
Occurrence:	If >= 3 <input type="button" value="v"/>
Event type:	ASR - Invalid Answer Limit <input type="button" value="v"/>
Object:	Agent Transfer on Recognition Problems
Continuation:	Proceed <input type="button" value="v"/>
Finish Tasks:	All <input type="button" value="v"/>

Setting the **Finish Tasks** field to *All* ensures that any task that happens to be open at this point will be terminated. And according to the event type, it will be assigned the status *Recognition failure*.

The same status, *Recognition failure*, will be applied to tasks that are automatically terminated by one of the following ASR-related event handlers: *ASR – Invalid Answer Limit*, *ASR – No Input*, *ASR – No Match*, *ASR – No Match (Voice)*, *ASR – No Match (DTMF)*, *ASR – Max Processing Time*, *ASR – Max Speech Duration*.

Note that you usually set *Finish Tasks = All* only in one, single place in the entire call flow definition. All other ASR-related event handlers typically only trigger a re-prompt (prompting the caller to try again, and maybe giving some additional advice on the expected caller action), but do not leave the current input state, so *Finish Tasks* stays with the default setting *None* for those.

Status: Incomplete - Back-end error

After most back-end errors, the call must either be gracefully terminated or transferred to an agent. As a result, any business task that might be active at that time must be terminated and flagged as failed. To achieve this, all you need to do is define an *Error – Connector* event handler and set its **Finish Tasks** field to *All*. This makes sure that any open task will be finished in case of a back-end error, with the status *Incomplete – Backend error*.

As an implementation best practice, it is recommended to define a generic *Error – Connector* event handler on the main module, which is then inherited by all Connector objects in the call flow (unless it is overridden by an *Error – Connector* event handler on a particular Connector object). On that generic event handler, you can make sure that the call is gracefully terminated and that business tasks are properly finished, whichever back-end connector fails.

Note that, sometimes, back-end errors are just a mere inconvenience rather than a showstopper. In those cases, an associated business task may still be completed – so, the **Finish Tasks** field needs to be set to *None* in this case.

Status: Incomplete - Technical error

Incomplete - Technical error is similar to the *Incomplete – Backend error* situation: This status will be automatically assigned in the context of event handlers of type *Error – Internal*, *Error – Media Platform*, and *Error – Script*, provided that *Finish Tasks = All* was set in those event handlers.



Status: Incomplete - Caller abort

The *Caller abort* completion status reflects pretty much any action a caller can take in order to proactively leave the context of a business task. The only exception to such actions is a caller hangup – this situation is captured by the *Session termination* task completion status (see above).

Here are some examples for business tasks failing due to *Caller abort* situations: The caller may ...

- Press “dtmf-0” to talk to an agent (“zero out”);
- Say “start over” or “back to main menu” to start over from scratch;
- After providing all details about some flight booking, when asked for final confirmation, the caller may say “no”, aborting the transaction.

In VoiceObjects, each of these interactions is typically modeled by a Hyperlink object that is embedded or linked in to the custom navigation of a Module or dialog component object. All you need to do, in terms of business task control, is this:

- Locate those hyperlinks in custom navigation that will force the dialog to leave the current task context (note that not all hyperlinks do that – they may also, for example, just trigger a re-prompt or help prompt, or switch some dialog parameter such as the current dialog language – all of which will not affect any ongoing business task completion). Typically, you can identify these hyperlinks because they call a Transfer or Exit object, and/or because the **Continuation** is set to *Do not return*.
- In those hyperlinks, set **Finish Tasks** to *All*, so VoiceObjects Server will always terminate any open business task and automatically set its status to *Caller abort* when the respective hyperlink is triggered.

The following image shows such a hyperlink definition in the context of custom navigation:

The image shows a configuration window titled "Custom Navigation (...)" with the instruction "Define the list of Hyperlink objects to be used as custom navigation commands." Below this is a "Hyperlink" configuration section. The "Destination (...)" sub-section contains the following fields:

Label:	Zero out
Layer:	
Channel:	Voice
Mode:	Process object
Object:	Agent Transfer
Event:	- Select or Enter Type -
Continuation:	Do not return
Finish Tasks:	All

Below the "Destination" section are sections for "Activation Grammar" and "Presentation Output". The "Finish Tasks" field is highlighted with a green border.



All these hyperlinks that indicate a *Caller abort* are typically generic – being available in all (or almost all) input states in an application –, and should hence be defined only once, in the start module of an application. Hence, there is a single place where to control their impact on business task completion.

There are a few other situations where, when triggered, you can set *Finish Tasks = All* causing VoiceObjects Server to terminate open tasks and mark them as *Incomplete - Caller abort*:

- Event handlers *Caller - Exit*, *Caller - Cancel*, *Caller - Help*
- Event handlers with some *Custom* type
- Goto objects

Status: Incomplete - Business logic

Sometimes, the call flows smoothly, speech recognition works like a charm, the caller is cooperative – and still, a task cannot be completed, for example because

- the caller entered a wrong PIN, or an invalid credit card number;
- the currency amount to be transferred exceeds the customer's limit; or
- no flight is available on that particular day, and so on.

In either of those cases, speech recognition and back-end access may have worked just fine, but some business logic stood in the way of task termination.

If this happens, you want to manually terminate the associated business task by calling the Expression function `FINISHTASK (task, "businessLogic"`).

Status: Incomplete - Task restart

When a task is started and VoiceObjects finds that another task associated with the same task object is still active, that other task is terminated with status *Incomplete – Task restart*.



Business Task Status - Summary

As a summary of the previous sections, the following table shows all possible scenarios for automatic task termination.

Condition		Task Status	Manual	Completion	
Automatic assignment of completion status					
Event handler	ASR (NoMatch, ...)	Recognition failure	(X)	Incomplete	
	Disconnect	Session termination	-		
	Error Connector	Backend error	(X)		
	Error [other]	Technical error	(X)		
	Caller Exit / Cancel				
Hyperlink		Caller abort	(X)		
Object	Goto, Exit				
<i>STARTTASK called again</i>		Task restart	-		
Manual assignment only					
<i>FINISHTASK("businessLogic")</i>		Business logic	X		
<i>FINISHTASK("complete")</i>		Complete	X	Complete	

To summarize, this is what needs to be taken care of in a VoiceObjects call flow implementation with business tasks:

- Define start and *successful* completion criteria by adding Expression objects to the call flow, using the STARTTASK and FINISHTASK functions.
- In all event handlers and hyperlinks that trigger a logical context switch in the dialog or a call transfer, set *Finish Tasks := All*. These event handlers and hyperlinks should typically be defined only once, in the start module of an application, and in a typical IVR call flow definition comprise
 - *Event handlers: ASR – Invalid Answer Limit(3), Error – Connector, Error – Internal, Error – Script, Error – Media Platform, Caller - Cancel, Caller - Exit*
 - *Hyperlinks: “Zero out”, “Start over”, ...*



5 Controlling Business Tasks: A Hands-on Example

As a practical example, let us consider 3 different options to design an “Authentication” business task around a simple loop prompting a caller for their PIN.

The loop in this call flow contains an input state, “Ask for PIN”, followed by a back-end check “Validate PIN”. Depending on the result, the flow proceeds after the loop (on successful authentication), or re-enters the loop.

After entering 3 invalid PINs in a row (PINs matching the grammar, but failing validation with the authentication back-end), the call flow breaks out of the loop and transfers the caller to an agent.

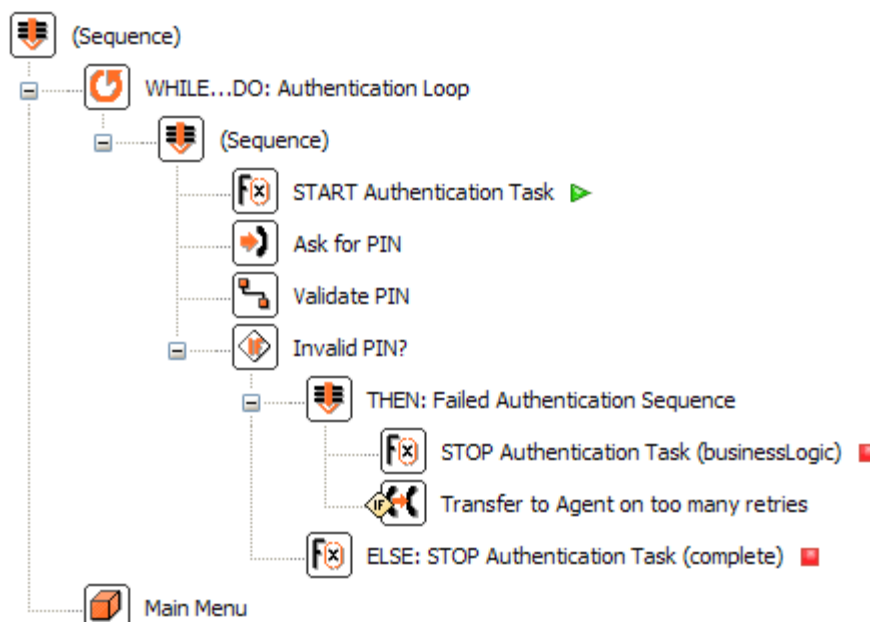
To measure the success of caller authentication, a Business Task object *PIN Authentication* was created. All 3 implementations will have this behavior in common:

- When the Connector *Validate PIN* fails, e.g. due to inaccessibility of the back-end system, an event of type *Error – Connector* will be triggered, and the *PIN Authentication* task will be terminated and marked as *Backend error*.
- When speech or DTMF recognition fails 3 times in a row (e.g. because the caller fails to enter a 4-digit PIN), an *ASR - Invalid Answer Limit* will be triggered, and the *PIN Authentication* task will be terminated and marked as *Recognition failure*.

Both the *Error – Connector* and the *ASR - Invalid Answer Limit* are implemented in the main Module object of the application (which is not displayed in the call flows below).

First Option

First, let us consider the situation where the *PIN Authentication* task is started and stopped *within* the Loop object. It's started right before the *Ask for PIN* input state, and finished as *Complete* in case validation succeeds. If validation fails, it is finished as *Incomplete – Business logic*. And after 3 such invalid attempts, the caller is transferred to an agent:





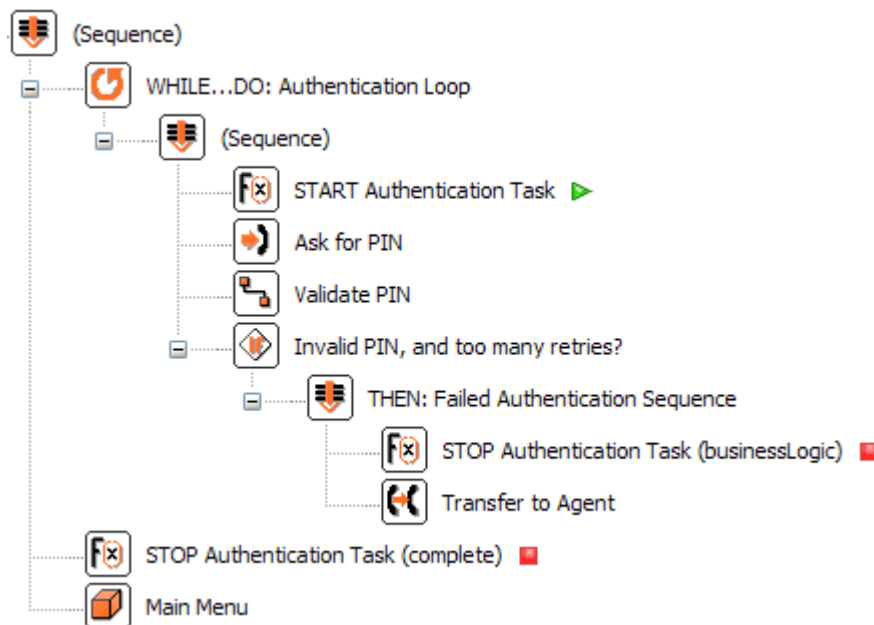
So, if a caller first enters a wrong PIN, but manages to enter the right PIN with the second attempt, the report will show *two* executions of the *PIN Authentication* business task; the first one as *Incomplete – Business logic*, the second one as *Complete*.

And if the caller enters three invalid PINs in a row, the report will show three task executions, all of which marked as *Incomplete – Business logic*. Hence, no distinction is made between a first and second invalid attempt – after which the caller may still be successful –, and the third invalid attempt – at which point we know that the caller has ultimately failed to authenticate.

So, let's see whether we can do better than that.

Second Option

Now, let us slightly change the implementation: The task is still started *within* the loop, but it is finished only after successful PIN entry and validation right *after* the loop, or after 3 invalid attempts (right before the agent transfer):



The business task reports will be different now.

First, re-consider the entry of an invalid PIN followed by a valid PIN entry. The report will again show *two* executions of the *PIN Authentication* business task: the first one as *Incomplete – Restart* (because the *PIN Authentication* task was restarted in the loop without having been finished before), the second one as *Complete*.

If 3 invalid PINs are entered in a row, you'll find 2 executions with status *Incomplete – Restart*, and a third execution with status *Incomplete – Business logic*.

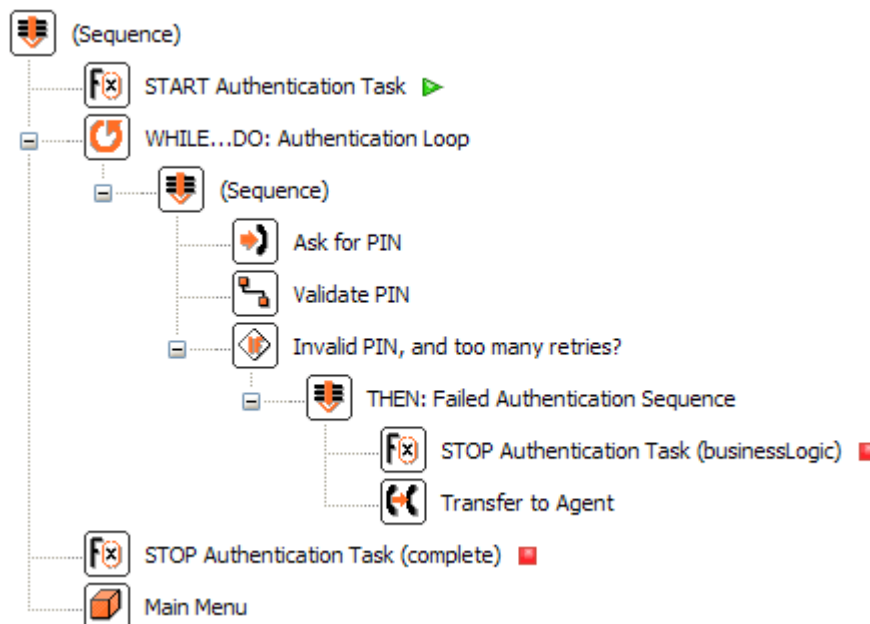
This is obviously a better implementation than the first: By just focusing on the *Complete* and the *Incomplete – Business logic* executions, you'll get data on the ultimate success rate of this task; and by looking at the numbers of tasks with status *Incomplete – Restart* and comparing that with the *Complete* count, you can derive the average number of retries required.



However, we can still do better.

Third (and recommended) Option

Let us do one final change: Move the expression that starts the *PIN Authentication* task out of the loop. The idea is now to treat the entire sub-dialog (including the loop) as a single task, involving one or several attempts at entering a valid PIN. In any given call, the task will hence only be finished once – either successfully, or with status *Incomplete – Business logic* in case of 3 subsequent invalid PIN entries.



Let us consider our two scenarios again: First, entry of an invalid PIN followed by a valid PIN entry. With this third implementation, the report will show *only one* execution of the *PIN Authentication* business task with status *Complete*. And for the other case – 3 invalid PINs entered in a row –, you will also find one single task execution with status *Incomplete – Business logic*.

This is comprehensive information, and directly shows the key performance indicator: The percentage of *PIN Authentication* task executions with status *Complete* gives the exact rate of callers who successfully authenticated in the IVR by entering their PIN.

Yet, you might still be interested in how often callers cycle through the loop until they enter a valid PIN. The good news is that this information is there, out-of-the-box, in Infostore. This is because in the context of a task execution, VoiceObjects Server automatically captures several counters and timers, including the count of input states, events, connector errors etc.

To get there, let us have a look at another task-related report in VoiceObjects Analyzer, the *Business Task Session Analysis* report.



The Business Task Session Analysis Report

The following screenshot from the *Business Task Session Analysis* standard report again shows data from using the *Prime Telecom* sample application. Focusing on the business task *Update Credit Card Expiration Date*, you see that successful attempts required exactly two input states on average (and if you know the application design, you'll recognize that these two states are the initial input state, followed by an explicit confirmation state).

You also get the information as to how many speech recognition related events had occurred in the context of that task, how many Connector objects were executed and how many of those failed, and a lot more information that is not displayed in this screenshot.

Task	Task Completion	# Tasks	%	# Sessions	# Task per S.	Task Dur. (active) (sec.)	Task Dur. (inactive) (sec.)	Avg Input States	% Succes.	Avg Input - No Match	Avg Input - No Input
Enter New Credit Card	<u>Complete</u>	1.300	61,9%	1.251	1,04	61,7	0,0	5,2	100,0%	0,8	0,3
	<u>Incomplete</u>	801	38,1%	788	1,02	40,0	0,0	3,3	74,8%	0,8	0,4
Order Tariff Addon	<u>Complete</u>	883	12,6%	786	1,12	9,1	5,3	1,0	100,0%	0,1	0,1
	<u>Incomplete</u>	6.111	87,4%	5.259	1,16	9,7	0,1	1,0	94,8%	0,1	0,1
PIN Authentication	<u>Complete</u>	7.640	88,9%	7.640	1,00	9,3	0,0	1,0	100,0%	0,1	0,1
	<u>Incomplete</u>	951	11,1%	931	1,02	11,1	0,0	1,0	36,0%	0,3	0,2
Update Credit Card Expiration Date	<u>Complete</u>	1.900	87,2%	1.791	1,06	18,3	0,0	2,0	100,0%	0,2	0,1
	<u>Incomplete</u>	280	12,8%	275	1,02	18,1	0,0	1,6	45,6%	0,4	0,3

You might have noticed that the values *Complete* and *Incomplete* in the column *Task Completion* are underlined. In the web interface of VoiceObjects Analyzer, you can click one of the *Incomplete* links in order to drill down into the different statuses for a given business task:

Task	Incomplete	# Tasks	%	# Sessions	# Task per S.	Task Dur. (active) (sec.)	Task Dur. (inactive) (sec.)	Avg Input States	% Succes.	Avg Input - No Match	Avg Input - No Input
Update Credit Card Expiration Date	<u>Caller abort</u>	37	13,2%	37	1,00	13,0	0,0	1,6	100,0%	0,1	0,1
	<u>Recognition failure</u>	96	34,3%	96	1,00	33,9	0,0	1,6	36,4%	1,0	0,7
	<u>Session termination</u>	147	52,5%	147	1,00	9,1	0,0	1,6	38,0%	0,0	0,0



6 Custom Logging with Business Task Data

There is one final, important feature in the Business Task object that we have not yet talked about: Business task data.

Using Business Task Data

Using business task data, any custom logging requirements can be met. The core idea is that pretty much any meaningful custom logging will happen in the context of some business task, for example

- Logging the credit card type in the context of credit card registration;
- logging the currency amount in the context of a money transfer;
- logging the code of the agent skill group in the context of a call routing; or
- logging the number of attempts a caller needed to authenticate with their PIN.

Adding custom logging to business tasks is easy: Drag and drop the Variable or Expression objects that you want to log into the parameter set of the respective task.

This is an example of the *Enter new Credit Card* business task in Prime Telecom; the variable *Credit Card Type* has been added to the set, and has been assigned the alias *CreditCardType*:

The screenshot shows the configuration interface for a business task. It is divided into several sections:

- Definition (...)**: A text box labeled "Name:" contains the text "Enter New Credit Card".
- Declaration (...)**: This section is currently empty.
- Parameter Sets (...)**: This section contains a descriptive text: "Specify the set of parameter objects and/or constants which will be associated with this task. Once this task is finished during a call, these parameters will be logged to Infostore." Below this text is a checkbox labeled "Use parameter set 'complete' for both sets", which is currently unchecked.
- Parameter Set - Complete (...)**: This section contains a descriptive text: "Specify the set of parameter objects and/or constants which will be logged to Infostore if the task was finished as 'complete'." Below this text is a sub-section for a parameter set:
 - CreditCardType**: A sub-section header.
 - Parameter:** A dropdown menu with "Credit Card Type" selected.
 - Alias:** A text box containing "CreditCardType".
- Parameter Set - Incomplete**: This section is currently empty.

This has the following effect: Whenever the task *Enter New Credit Card* is finished in the call flow – regardless of whether it's finished manually or automatically – VoiceObjects Server not only writes the task completion status and all associated metrics to Infostore, but also takes a snapshot of the



current value of all parameters in the set. These values are written to Infostore as key/value pairs (with the *Alias* defining the key) along with this task execution result.

Note that there are two distinct parameter sets: One that defines the parameters that are to be persisted in case of successful task completion, and another one in case of task execution failure. If you don't want to distinguish between these two scenarios, select the **Use parameter set "complete" for both sets** checkbox.

When running the *Business Task Data* standard report in VoiceObjects Analyzer, you will be prompted for the service and for the task to report about:

VoiceObjects Analyzer

Business Task Data

Date

* Year (All)

[Deselect all](#)

Service

* Prime Telecom

Task

* Task

- Task
-
- Change Bank Details
- Change Email Address
- Change Postal Address
- Enter New Credit Card**
- Order Tariff Addon
- PIN Authentication
- Update Credit Card Expiration Date

Cancel < Back Next > Finish

In case of alphanumeric business task data, the *Business Task Data* report will show the absolute and relative frequency of each possible parameter value:

VoiceObjects Analyzer

Business Task Data

Task Completion	Task Data Key	Task Data Value	Count	% by Data Key
Complete	CreditCardType	AmEx	366	18,5%
		MC	894	45,1%
		Visa	722	36,4%

Report Filter

Date: *Year (All)*

Service: *Prime Telecom*

Task: *Enter New Credit Card*



And in case of numerical data, the report will present aggregated data including the total sum, min, max, and average values of the parameter value:

VoiceObjects Analyzer

Business Task Data

Task Completion	Task Data Key	Total	Count	Count Distinct	Min	Max	Avg
Complete	AddonMonthlyRate	15,778.77	1,123	3	9.99	39.99	14.051

Report Filter

Date: *Year (All)*
Service: *Prime Telecom*
Task: *Order Tariff Addon*



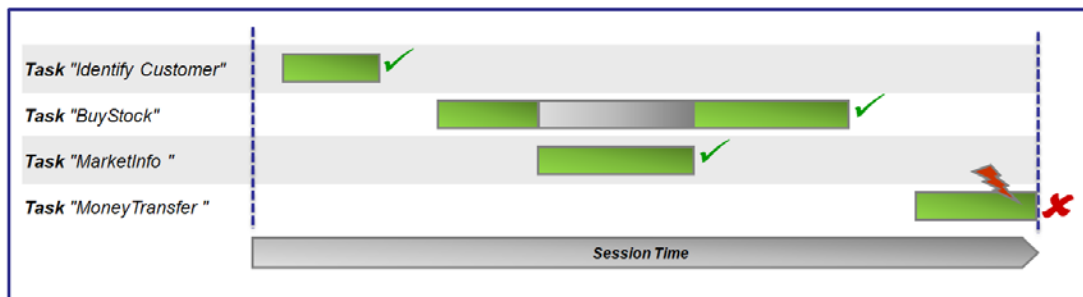
7 Wrapping Up

In this final section, we want to wrap up by discussing some additional business task related features that we have skipped so far.

Stacking Tasks

In an earlier section, we stated that as a best practice you should not stack business task executions on top of each other. It's always cleaner to finish the current task and then to start the next.

However, sometimes it does make business sense to stack tasks. Consider the following call scenario, illustrated by a chart of task activities during the lifetime of the call session.



- Right after the start of call, the caller is successfully identified and authenticated by PIN. The business task *Identify Customer* is finished as *complete*.
- Now, the caller selects “stock trading”. The task *BuyStock* is started. After identifying the stock name, the caller requests current market information on that stock.
- This gets him into a different subdialog where a new task *MarketInfo* is duly started. This one is now the *active* task, while VoiceObjects Server made *BuyStock* inactive during this time.
- Once the caller successfully got the market information, the *MarketInfo* task is finished. VoiceObjects Server automatically re-activates the *BuyStock* task. The caller successfully resumes that dialog, eventually placing a “Buy” order.
- Back in the main menu, the caller starts a money transfer. Because of a back-end error, the task *MoneyTransfer* cannot be completed and the call is gracefully terminated by the system. The business task *MoneyTransfer* will be automatically flagged as *Backend error*.

So, while the task *MarketInfo* was active, the *BuyStock* task was inactive. This is handled automatically by VoiceObjects Server. When a task is inactive, metrics and counters for this task are also frozen; for example, input state level statistics such as count of input states, count of NoInput/NoMatch events, count of back-end connector executions and more will be attached only to the active task.

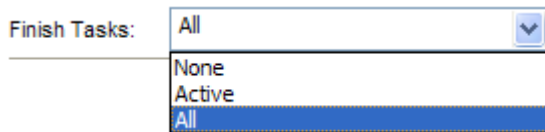


Reporting on stacked tasks

Looking back at the report image in *The Business Task Session Analysis Report*, you'll now understand the columns *Task Duration (active)* and *Task Duration (passive)*. Also note that the values in columns *Avg Input States*, *% Success* etc. only apply to the active phases of the task.

Finish task options

When setting the *Finish Tasks* value in event handlers, hyperlinks, and Goto objects, you actually have three options: *None*, *Active*, and *All*.



When selecting *Active*, only the task that is active (if any) at the time of this event handler (or hyperlink, or Goto object) being triggered will be finished. Tasks that are inactive are not affected. In consequence, the topmost task on the stack of inactive tasks will be re-activated.

When selecting *Any*, all unfinished tasks – both active and inactive – are being finished. In most situations, this is what you want: When the caller hangs up, presses “0” to talk to an agent, says “start over” to get back to the main menu, or when a fatal back-end error occurs, any task that was started (and not yet terminated) within this session must be considered as failed.

That's why we have always pointed at setting *Finish Tasks* = *All* in the sections above; setting *Finish Tasks* = *Active* will be required only under very specific circumstances.

Let's consider, once more, the call flow discussed in “Stacking Tasks”. In particular, let's think about the business task *MarketInfo* that was stacked on top of the *BuyStock* task.

In event handlers pertinent to the context of the *MarketInfo* task, you would set

- *Finish Tasks* = *Active* if triggering this event handler would cancel or interrupt the *MarketInfo* task, giving back control to the *BuyStock* call flow where the *BuyStock* task can be successfully completed;
- *Finish Tasks* = *All* if triggering this event handler would break both the *MarketInfo* and the *BuyStock* tasks at the same time, e.g. because it provoked an agent transfer, system hangup, fallback to the main menu, or something alike.

Stacking tasks – best practices

In the example we presented in *Stacking Tasks*, it made sense to start a second task *MarketInfo* while the task *BuyStock* was still active, because that first task was interrupted – but not cancelled – by the caller's intent to get more information before getting back to buying the stock. Input state related metrics will be appropriately associated with each of these two tasks.

We advise against stacking tasks, though, for the sole purpose of sub-dividing a full task into smaller sub-tasks. The problem with this approach is the following: You won't get meaningful input-state level stats for the full task (it will remain inactive for most of the time). Also, there is the risk that the sub-tasks won't provide any more insight on top of the input-state level information that is collected by VoiceObjects Server anyway – you might end up generating much more data in the Infostore database without adding additional insight.



If you have a complex task consisting of several input states, connectors, and more, you may want to have an easy way of finding out where things might break for callers. What about this: Create a status parameter that is initialized on task start, and incremented or re-assigned after each successful input state. Add that status parameter as task data to the business task's parameter set. Thus, the *Business Task Data* report will automatically provide statistics on bottlenecks in the call flow related to your complex task.

Business Task Expressions

Apart from the two *STARTTASK* and *FINISHTASK* expression functions, there are two additional expressions that come in handy when you need to implement call flow logic that depends on task status, or on which task is currently active:

- **TASKREFID ()**: Returns the reference ID of the currently active business task. If no task is active an empty string is returned.
- **TASKSTATUS (task)**: Returns the most recent status of the business task specified in the first argument. If the task has not been finished yet, it returns *active* or *inactive*. If the task has never been started, it returns *notStarted*.

Input State-Level Reports with Business Tasks

The reports that we inspected so far centered around business tasks. However, business task related information can also be added to other reports. For example, it is an easy exercise to add the column *Task* to a report that shows input state details for a single session:

Input State Step	Task	Input State	Utterance	# No Input Events	# No Match Events	Avg Confidence	External Duration (sec.)
0		Main Menu	customer data	0	0	0.610	19
1	PIN Authentication	Get PIN	4629a38106ba14ad7	0	0	0.630	17
2		Customer Data	credit card	0	0	0.630	10
3		Credit Card	enter new card	0	0	0.710	19
4	Enter New Credit Card	Get Credit Card Type	american express	0	0	0.530	9
5	Enter New Credit Card	Get Credit Card Number	68f7610ae31dd9610:	0	0	0.000	7

For each input state, you'll get the information as to which task was active at that time. This will be very useful for verifying your call flow and general troubleshooting.



8 Further Information

You can find a number of additional sources for further information on business tasks in VoiceObjects.

Documentation

The Business Task object is documented in the *Object Reference* in the VoiceObjects documentation that is available both in your local VoiceObjects Desktop for Eclipse installation and online on the web:

<http://developers.voiceobjects.com/docs/en/VO10/index.htm#004-businessstask.htm>

Jam Session

On the VoiceObjects Jam Session archive page, <http://developers.voiceobjects.com/tech-topics/monthly-jam-sessions/>, the session *October 07, 2010 – Voxeo VoiceObjects – Reporting and Analytics Best Practices* provides lots of information about reporting for VoiceObjects developers. You'll find slides and a recording of the webinar.

Sample Applications

The two VoiceObjects standard sample applications, *Prime Insurance* and *Prime Telecom*, have implemented business tasks. You can inspect those implementations to see how it can be done.